

Как учить большие языковые модели

Апрель, 2024



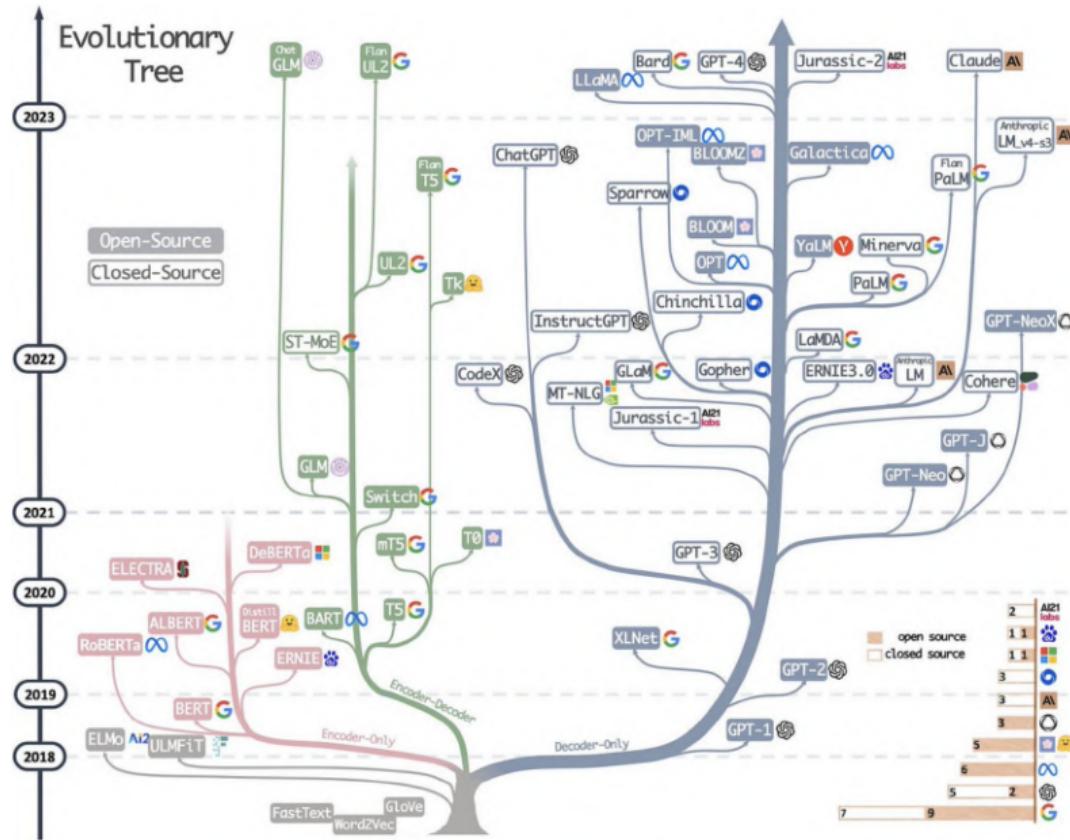
Мурат Апишев

Search Tech Lead, Samokat.Tech
ex-Lead Data Scientist, SberDevices
mel-lain@yandex.ru

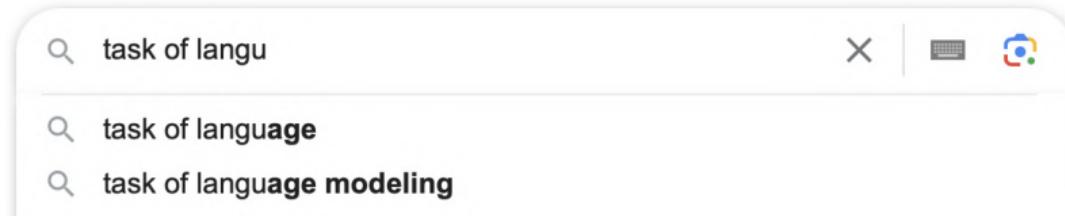
Сейчас «LLM» ≈ «AI»

Современные AI-сервисы:

- ▶ общаются с людьми на любые темы на естественном языке
 - ▶ решают школьные и университетские задачи по разным дисциплинам
 - ▶ понимают и генерируют тексты, изображения, аудио



Языковое моделирование



- ▶ У задачи моделирования языка есть две эквивалентные постановки:
 - ▶ предсказать совместную вероятность произвольной последовательности слов из n
 - ▶ предсказать вероятность следующего слова после произвольной последовательности из $n - 1$ слов
- ▶ LLM учатся на корпусах текстов задаче предсказания следующего слова по предшествующему контексту
- ▶ Иногда контекст оказывается не только из прошлого, а из будущего (Fill-in-the-Middle), например, для задач code infilling

Промпting LLM

- ▶ Современные большие языковые модели общего назначения:
 - ▶ мультиязычные
 - ▶ мультидоменные
 - ▶ инструктивные
- ▶ Благодаря этому они могут решать разные задачи, получая их на естественном языке (*prompt*), как исполнитель-человек
- ▶ Поставить задачу можно разными способами, стандартные подходы:
 - ▶ **Zero-shot:**

«Вопрос: у Васи 4 яблока, у Пети 8 груш, сколько у них съел половину своих фруктов, сколько всего фруктов осталось? Ответ: »
 - ▶ **Few-shot:**

«У Лены было 3 конфеты в одной руке, и 5 в другой, 2 она отдала Маше. Сколько конфет всего осталось у Лены? Ответ: 6
Вопрос: у Васи 4 яблока, у Пети 8 груш, сколько у них съел половину своих фруктов, сколько всего фруктов осталось? Ответ: »

Промпting LLM

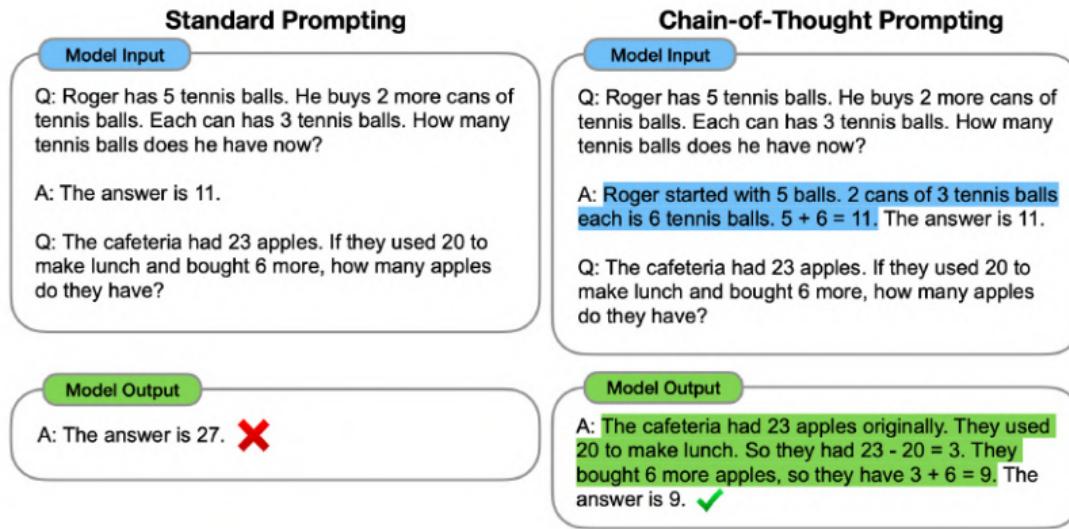
- ▶ Поставить задачу можно разными способами, стандартные подходы:
 - ▶ **Chain-of-Thought** (может быть и с zero-shot, и с few-shot):

«У Лены было 3 конфеты в одной руке, и 5 в другой, 2 она отдала Маше. Сколько конфет всего осталось у Лены? Давай порассуждаем: всего у Маши $3 + 5 = 8$ конфет. $8 - 2$ равно 6. Ответ: 6

Вопрос: у Васи 4 яблока, у Пети 8 груш, сколько у них съел половину своих фруктов, сколько всего фруктов осталось? Давай порассуждаем: »
- ▶ В этом случае модель сперва сгенерирует промежуточные рассуждения, а затем, на их основании, ответ
- ▶ Модель будет хорошо решать задачи с разными типами промптов, если она этому обучалась
- ▶ Т.е. чтобы модель умела рассуждать последовательно, она должна видеть в данных много примеров таких рассуждений

Промпting LLM

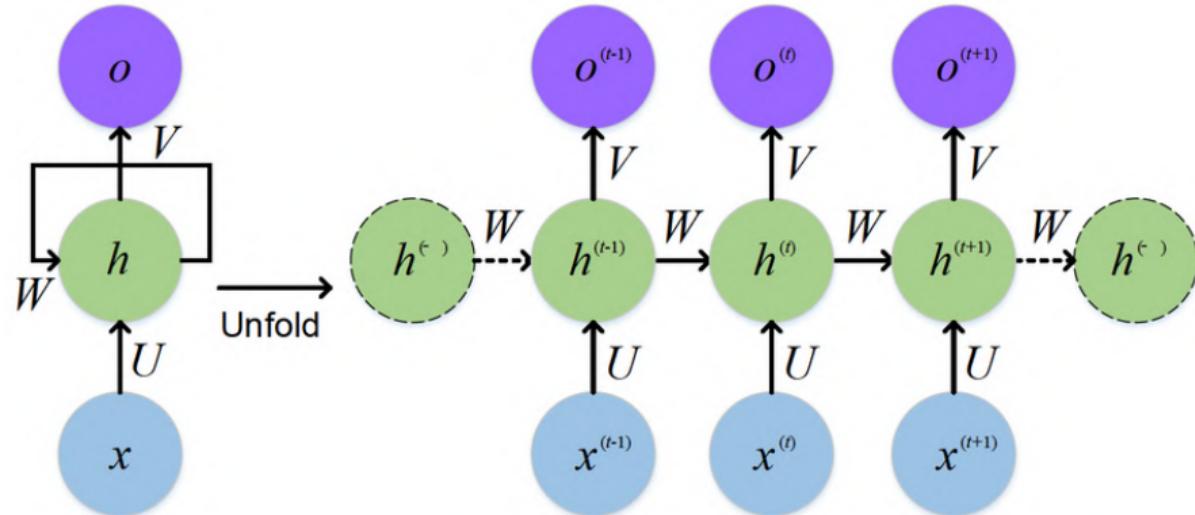
- ▶ Подбор промпта сильно влияет на качество ответов, ещё пример с CoT:



- ▶ Авторы учат модель как можно более широкому пониманию инструкций, пользователи перебором ищут лучшие варианты для своих задач
- ▶ Иногда промпты пытаются настраивать автоматически на целевую задачу (в том числе как замену дообучению, p-tuning)

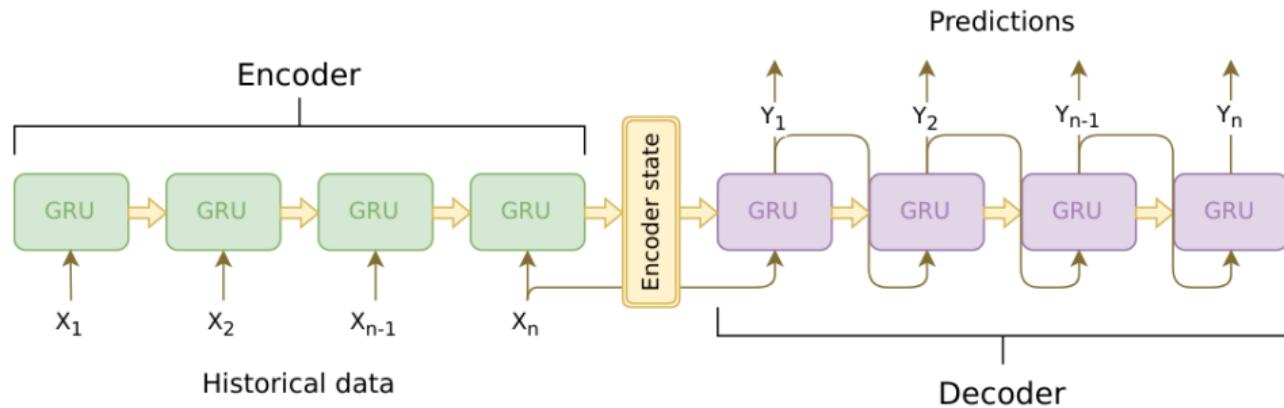
Архитектура

- ▶ До LLM: рекурентные нейросети (RNN), а именно LSTM, GRU, MGU
- ▶ Обрабатывают слово за словом, передавая обновляемый вектор состояния (и иногда дополнительный вектор) с информацией о последовательности
- ▶ Модель состоит из нескольких обучаемых весовых матриц



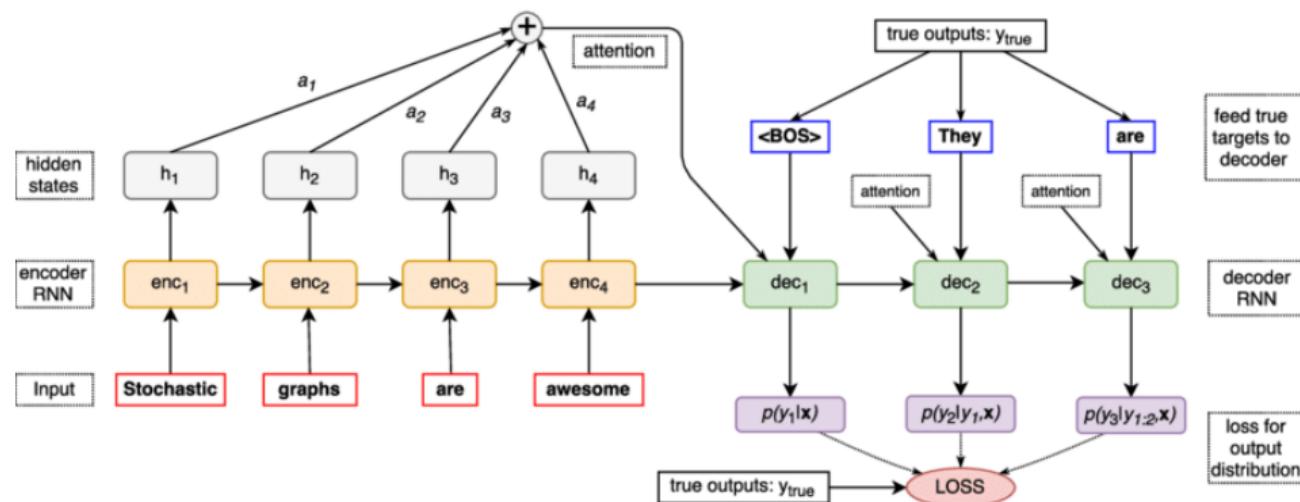
Архитектура

- ▶ RNN можно использовать для задач sequence-to-sequence (например, перевод или суммаризация)
- ▶ Нейросеть-кодировщик генерирует вектор состояния по входу
- ▶ Нейросеть-декодировщик генерирует по нему выход
- ▶ Работает не очень хорошо — на длинных последовательностях вектор теряет информацию о словах в начале



Архитектура

- ▶ Для борьбы с забыванием добавляется механизм внимания
- ▶ Декодировщик при генерации очередного слова определяет важность каждого из слов входа и использует взвешенную сумму выходных векторов кодировщика как дополнительную информацию
- ▶ Качество сильно выросло, это был стандарт в области

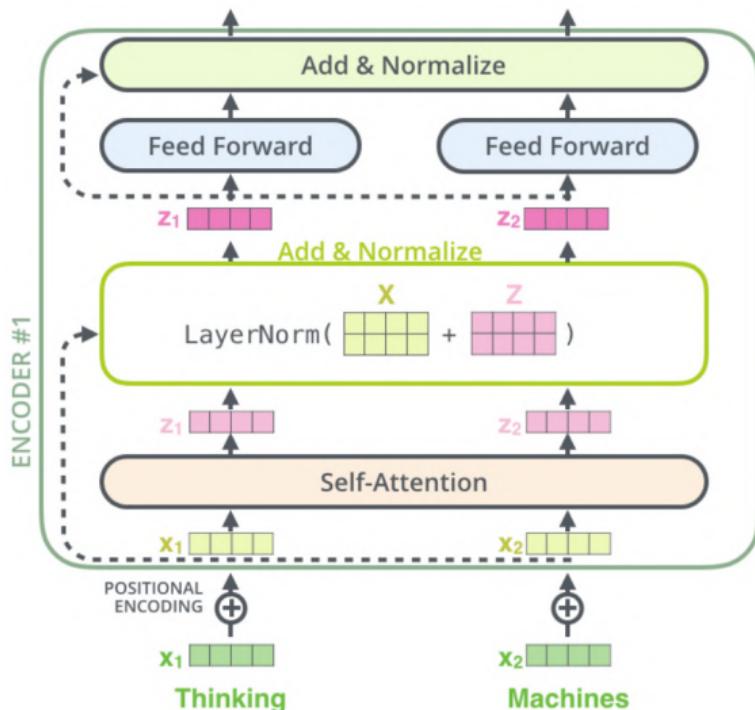


Архитектура

- ▶ Можно формировать векторы слов с учётом того, какие слова ещё есть в последовательности — получим Self-Attention
- ▶ Общая схема:
 - ▶ каждому слову входа сопоставляется вектор (на входе — эмбеддинг)
 - ▶ с помощью трёх обучаемых весовых матриц он переводится в три новых вектора: запросы, ключи и значения
 - ▶ новым представлением для слова будет взвешенная сумма всех векторов значений, нужно только определить, какие прочие слова для целевого слова важны
 - ▶ для этого запрос слова умножается скалярно на все ключи
 - ▶ после нормирования Softmax эти значения становятся весами суммы
- ▶ Можно считать несколько Self-Attention с разными весами конкатенировать результаты — Multi-Head Attention
- ▶ Нужно только добавить линейный слой проекции в исходную размерность

Архитектура

- Добавляются два полно связных слоя, Layer-нормализация и residual connection и получается блок Transformer:

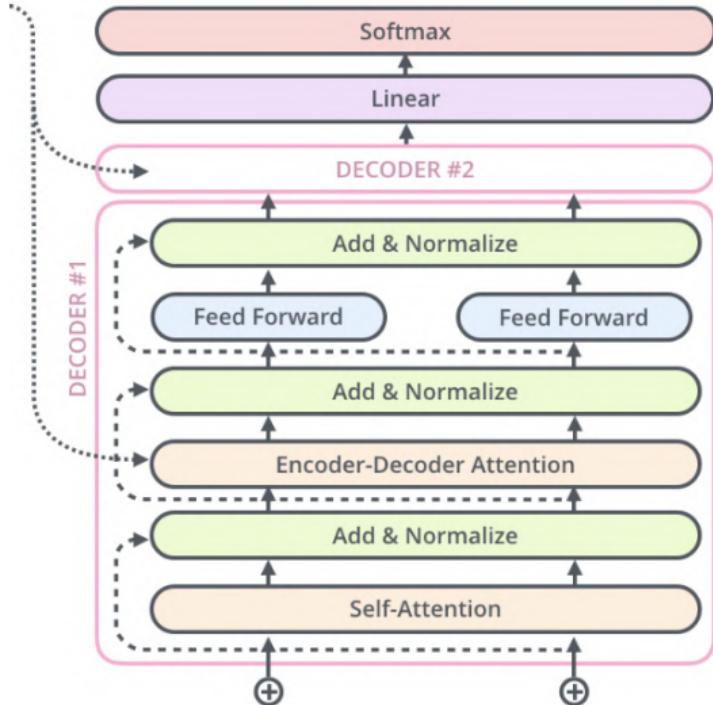


Архитектура

- ▶ Если поставить друг на друга несколько таких блоков, получится кодировщик Transformer
- ▶ Векторы слов многократно проходят через self-attention, на выходе получаются качественные контекстнозависимые представления
- ▶ Их уже можно использовать для решения разных задач напрямую или после дообучения небольшой модели-голов
- ▶ На основе архитектуры кодировщика обучены модели типа BERT
- ▶ Но оригинальный Transformer — sequence-to-sequence
- ▶ Нужен ещё и декодировщик, который на основе выходов кодировщика будет генерировать выходной текст

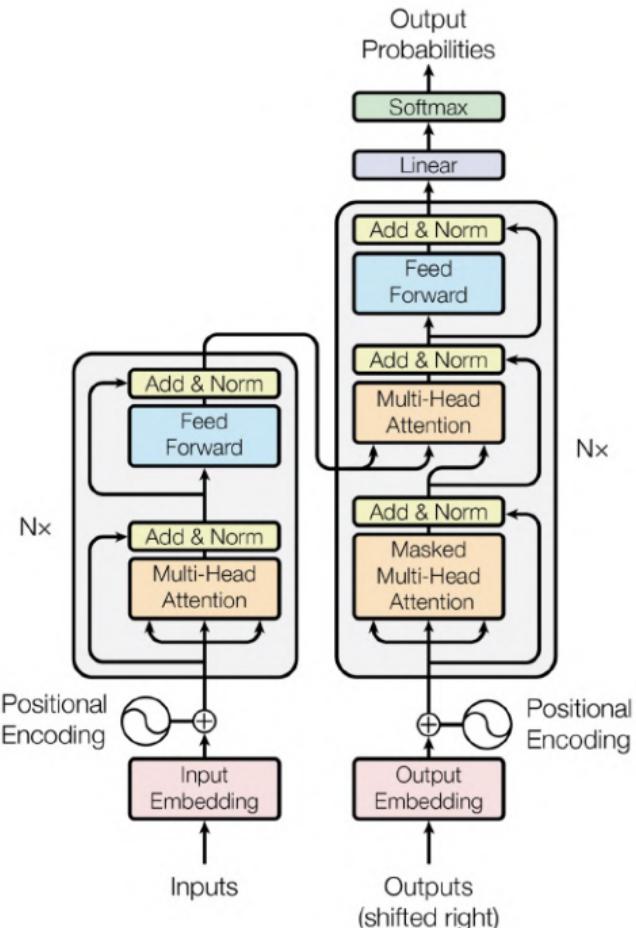
Архитектура

- ▶ Блок декодировщика похож на блок кодировщика, но есть отличия:
 - ▶ Masked Self-Attention для авторегрессионности
 - ▶ Cross-Attention между векторами сгенерированных и входных слов
 - ▶ запросы получаются из сгенерированных, ключи и значения — из входных
 - ▶ на выходе всего декодировщика softmax для генерации (как и в RNN)



Архитектура

- ▶ Кодировщик с декодировщиком образуют полный Transformer (модели типа T5)
- ▶ Декодировщик часто используют отдельно (модели типа GPT)
- ▶ Чаще всего LLM обучаются на основе декодировщика Transformer (GPT, LLaMA, Qwen, Mistral, ...)
- ▶ За последние годы предложен ряд успешных архитектурных модификаций (pre-LayerNorm, RMSNorm, SwiGLU), но суть сохранилась



Токенизация

- ▶ На самом деле Transformer не работает со словами:
 - ▶ требуется очень большой словарь
 - ▶ сложность учёта морфологии
 - ▶ проблема OOV слов
- ▶ Модели на символах тоже непопулярны — слишком длинная последовательность и низкое качество
- ▶ Вместо этого используются токены — под слова, т.е. символьные N-граммы (среди которых могут быть и частотные слова целиком)
- ▶ Разбиение на токены производит токенизатор, он обучается статистически по текстам выбранным алгоритмом:
 - ▶ BPE
 - ▶ Unigram
 - ▶ Wordpiece

Токенизация

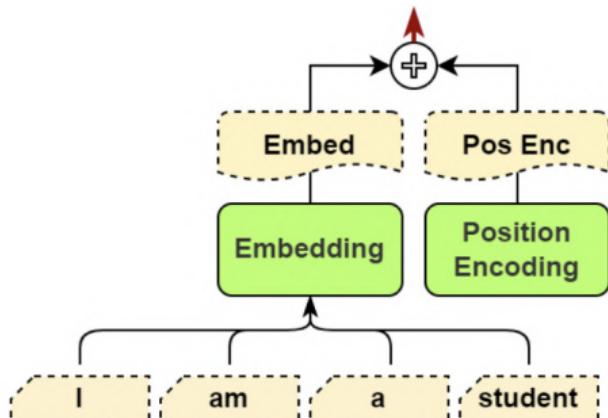
- ▶ Токенизатор разбивает текст на токены и сопоставляет каждому его номер

```
Two annoying things about OpenAI's tokenizer playground: (1) it's
capped at 50k characters, and (2) it doesn't support GPT-4 or GPT-3.5
...
So, I built my own version w/ Transformers.js! It can tokenize the
entire "Great Gatsby" (269k chars) in 200ms! ⏱️⏱️⏱️
```

- ▶ Для обработки символов, не встречавшихся в обучающих данных токенизатора, используется Byte Fallback — в словарь добавляются сразу все возможных 256 байтов
- ▶ Есть эксперименты по использованию чисто byte-level токенизаторов
- ▶ Ещё пробуют строить словарь на основе морфологии
- ▶ Рецепта создания идеального токенизатора пока нет

Позиционное кодирование

- ▶ Без дополнительной информации о позициях токенов любая модель на основе Transformer работает плохо
- ▶ В оригинальной реализации для позиционного кодирования
 - ▶ каждой позиции i токена сопоставляется вектор, содержащий различные значения синусов и косинусов от i
 - ▶ этот вектор добавляется к вектору эмбеддинга токена на позиции i перед отправкой в модель
- ▶ Способ простой и рабочий, но есть проблемы:
 - ▶ низкое качество кодирования \Rightarrow хуже результаты
 - ▶ низкое качество экстраполяции \Rightarrow нет обобщения на больший контекст

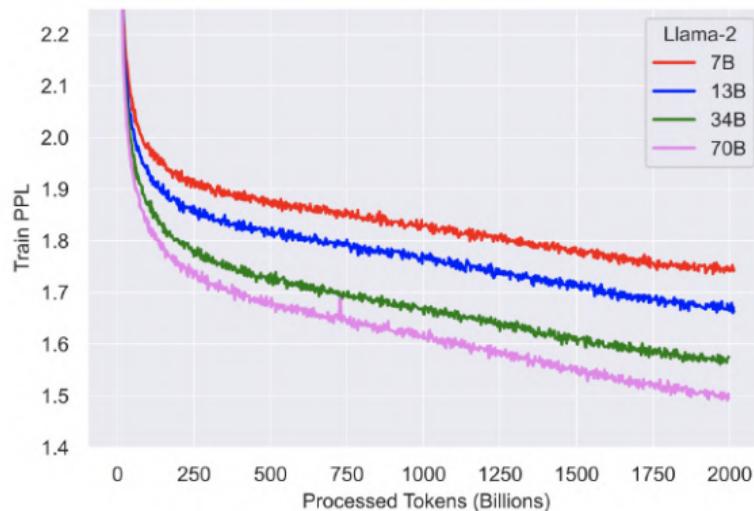


Позиционное кодирование

- ▶ Все более новые подходы — относительные, вместо позиции токена кодируется расстояние между парой токенов
- ▶ Вместо входного эмбеддинга модифицируется подсчёт self-attention, условно модификации можно разделить на три вида:
 - ▶ репараметризация формулы подсчёта внимания
 - ▶ Transformer-XL, 2019
 - ▶ DeBERTa, 2021
 - ▶ обычная формула с добавлением обучаемого сдвига
 - ▶ T5, 2020
 - ▶ AliBi, 2022
 - ▶ ротационное кодирование (RoPE, 2021) и его вариации
 - ▶ xPos, 2023
 - ▶ Positional Interpolation RoPE, 2023
 - ▶ YaRN, 2023
- ▶ Пробовали и вообще обходиться без кодирования позиций в декодерах (NoPE, 2023), но работает не очень

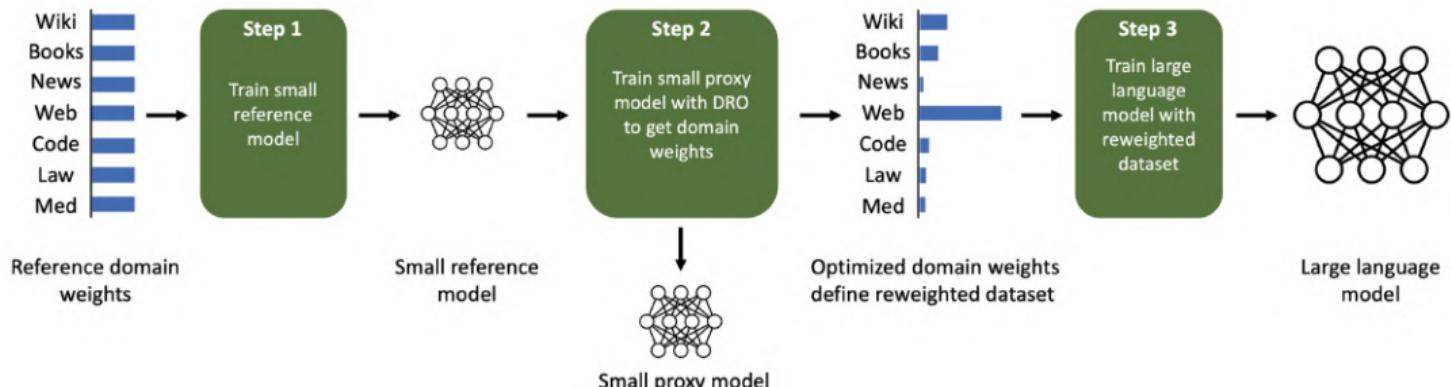
Этапы обучения

- ▶ Первая стадия обучения LLM — предобучение (pre-train)
- ▶ Модель учится предсказывать следующий токен по контексту слева
- ▶ Если учить с Teacher Forcing — контекст берётся из обучения, если без — из того, что сгенерировала в процессе сама модель (комбинируют)
- ▶ На этом этапе приобретает основные знания о языке и мире



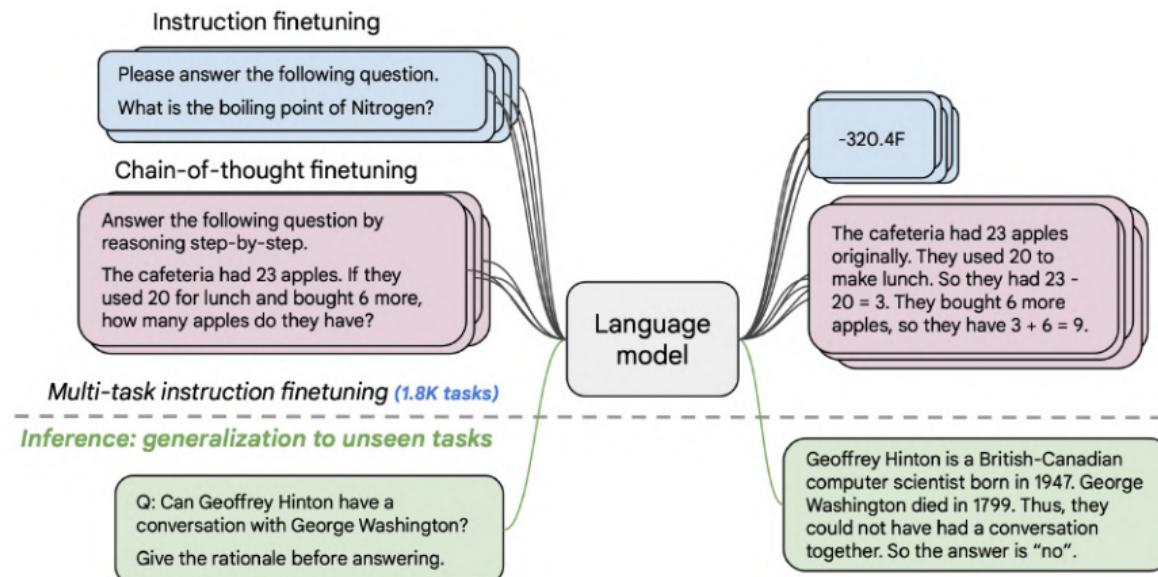
Данные для обучения

- ▶ Для обучения сильной LLM требуются терабайты данных, это много
- ▶ Но всё равно меньше, чем данных существует в природе
- ▶ И не все данные одинаково полезны, какие-то вредны (их фильтруют), какие-то ничего не приносят, но тратят вычисления
- ▶ DoReMi — пример того, как путём грамотного перевзвешивания доменов сильно уменьшить объём данных и вычислений с улучшением качества модели на few-shot задачах:



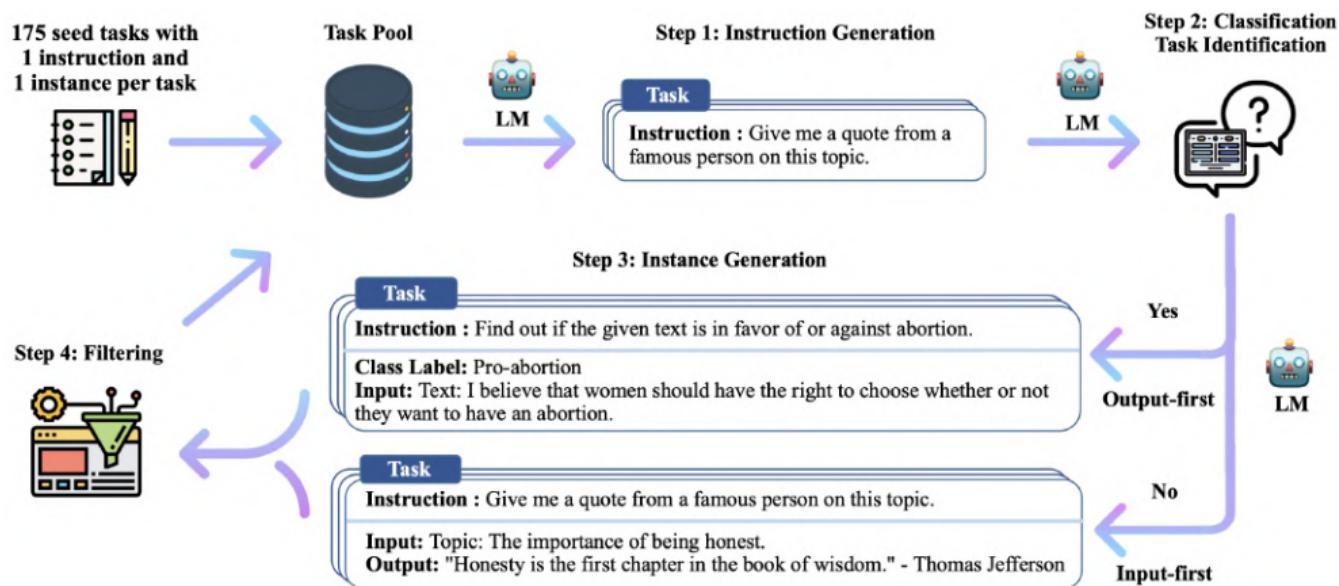
Этапы обучения

- ▶ Вторая стадия — Instruction Tuning (SFT)
- ▶ Модель учится понимать и исполнять запросы людей на естественном языке и вести диалоги
- ▶ Например: pre-train — LLaMA, instruct-tuned — Alpaca или Vicuna



Данные для обучения

- ▶ По сути модель так же обучается предсказывать следующий токен по прошлым, но на данных специального вида
- ▶ Объем данных сильно меньше, но требования к качеству высокие
- ▶ Сбор инструктивных датасетов сложен и дорог, пробуют Self-Instruct:



Данные для обучения

- ▶ Работа над данными и на претрейне, и на SFT позволяет сократить их объём и повысить качество, пример — кодовая модель phi-1 (и далее)
- ▶ Размеры относительно малы (до 1.3B), а качество сопоставимое с моделями в несколько раз больше
- ▶ Обучение на фильтрованных и синтетических данных, дообучение на качественных кодовых данных в стиле учебников

High educational value

```
import torch
import torch.nn.functional as F

def normalize(x, axis=-1):
    """Performs L2-Norm."""
    num = x
    denom = torch.norm(x, 2, axis, keepdim=True)
    .expand_as(x) + 1e-12
    return num / denom

def euclidean_dist(x, y):
    """Computes Euclidean distance."""
    m, n = x.size(0), y.size(0)
    xx = torch.pow(x, 2).sum(1, keepdim=True).
        expand(m, n)
    yy = torch.pow(y, 2).sum(1, keepdim=True).
        expand(m, m).t()
    dist = xx + yy - 2 * torch.matmul(x, y.t())
    dist = dist.clamp(min=1e-12).sqrt()
    return dist
```

Low educational value

```
import re
import typing
...

class Default(object):
    def __init__(self, vim: Nvim) -> None:
        self._vim = vim
        self._denite: typing.Optional[SyncParent] = None
        self._selected_candidates: typing.List[int] = []
        self._candidates: Candidates = []
        self._cursor = 0
        self._entire_len = 0
        self._result: typing.List[typing.Any] = []
        self._context: UserContext = {}
        self._bufnr = -1
        self._winid = -1
        self._winrestcmd = ''
        self._initialized = False
```

Данные для обучения

- ▶ Почти все знания модель получает на этапе предобучения
- ▶ Идея LIMA:
 - ▶ на SFT не нужно вкладывать в модель новую информацию
 - ▶ нужно как можно лучше объяснить модели, как общаться
- ▶ Для этого данных должно быть немного, но очень высокого качества
- ▶ Собранный вручную набор из 1000 примеров для LLaMA 65B позволил обучить модель высокого качества
- ▶ Гипотеза: хорошую SFT-модель не нужно выравнивать (будет дальше)

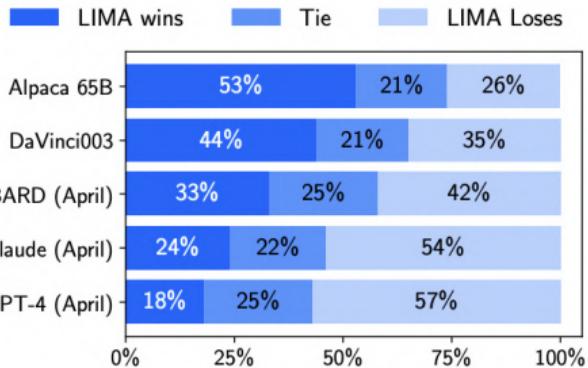


Figure 1: Human preference evaluation, comparing LIMA to 5 different baselines across 300 test prompts.

Данные для обучения

- ▶ Небольшие модели часто учат на выходах больших, и они часто вместо повторения рассуждений просто повторяют стиль
- ▶ В [Orca](#) предлагается дообучить LLaMA 13B на большом объёме специально собранных синтетических данных:
 - ▶ разнообразные задания и инструкции набираются из данных [FLAN](#), 2021
 - ▶ модель-учитель получает их на вход с дополнительными инструкциями, требующими детального объяснения ответа, например:

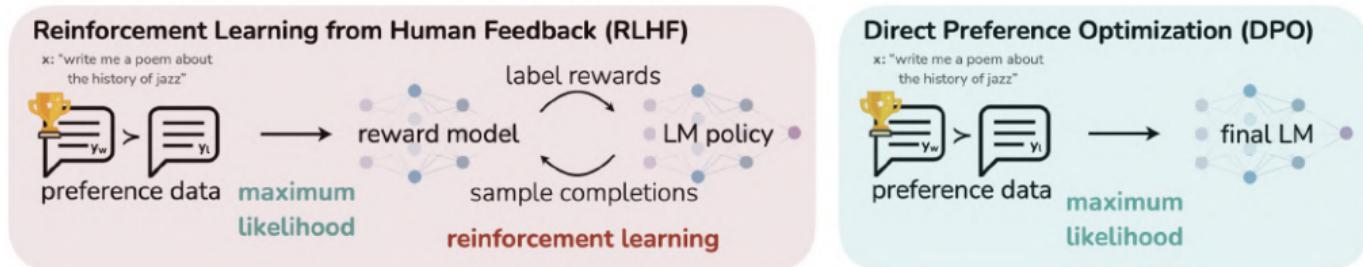
«*You should describe the task and explain your answer. While answering a multiple choice question, first output the correct answer(s). Then explain why other answers are wrong. Think like you are answering to a five year old.*»
 - ▶ модели генерируют ответы с объяснениями
- ▶ Модель-ученик тренируется на полученных тройках «системный промпт»-«задание»-«полный ответ учителя»
- ▶ Разнообразие и качество данных дополняется количеством: 1M сэмплов на основе GPT-4 и 5M — на основе ChatGPT

Этапы обучения

- ▶ Третий, optionalный, шаг — выравнивание (alignment)
- ▶ Диалоговая модель дообучается для генерации более корректных, полезных и безопасных ответов
- ▶ Популярная техника, использованная в Instruct GPT — RLHF:
 - ▶ обученная LLM генерирует на тестовом наборе инструкций ответы
 - ▶ ответы размечаются ассессорами, на их ответах учится сильная reward-модель, она оценивает по тексту его качество
 - ▶ заводятся две копии модели (A) и (B), учится (A)
 - ▶ обе модели генерируют ответы на каждый промпт, ответ (A) оценивается reward-моделью
 - ▶ веса (A) обновляются так, чтобы максимизировать reward и не давать ответы, очень далёкие от исходной (B)
 - ▶ расстояние определяется по KL-дивергенции между выходными распределениями моделей
 - ▶ обновление весов идёт по заданному алгоритму (PPO или A2C)

Этапы обучения

- RLHF сложен и не всегда работает хорошо, альтернатива — DPO:



- DPO устраняет необходимость обучения отдельной reward-модели и онлайн-генерации с RL
- Вместо этого функция потерь LM переопределяется и оптимизируется напрямую
- Она учитывает как предпочтения ответов из набора данных, так и требование не уводить ответы далеко от исходной модели
- DPO использует только бинарные оценки предпочтений, иные форматы нужно сводить к этому

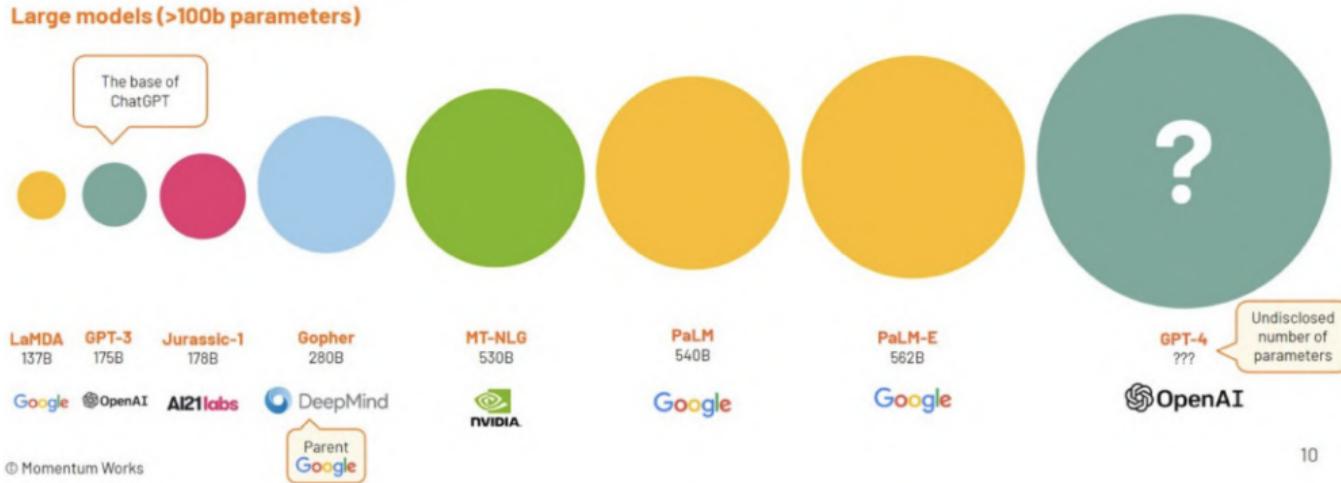
Баланс между параметрами и данными

- В общем случае модель чем больше, тем лучше, но только если обучена на достаточном объёме данных достаточное время

Small models (<= 100b parameters)

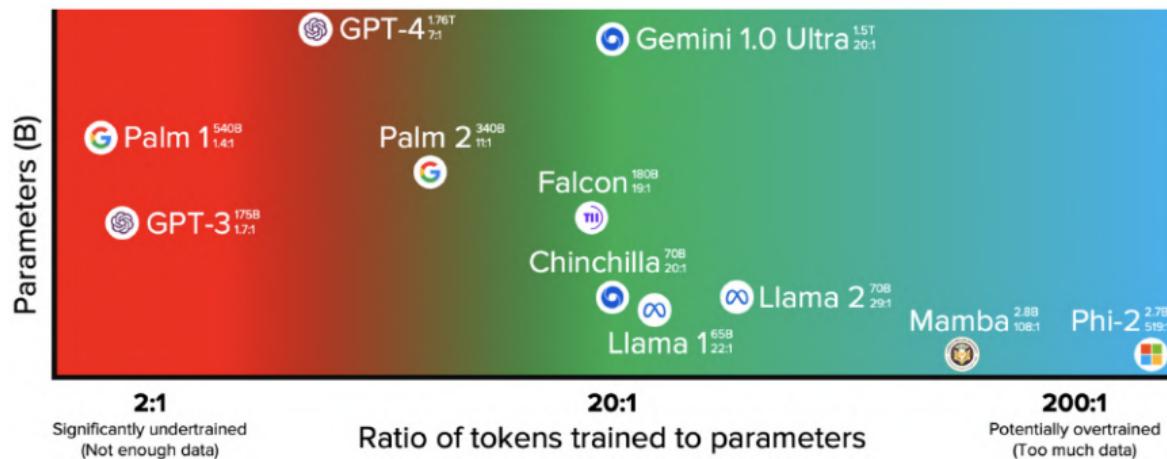


Large models (>100b parameters)



Баланс между параметрами и данными

- ▶ Авторы Chinchilla, 2022 задумались о важности баланса между размерами модели и объёмом данных (длительностью обучения)
- ▶ На обучение модели выделяется ограниченный вычислительный бюджет
- ▶ Его можно тратить, увеличивая либо размер модели, либо длительность её обучения, нужно балансируя loss
- ▶ Большинство моделей оказались обученными неоптимально:

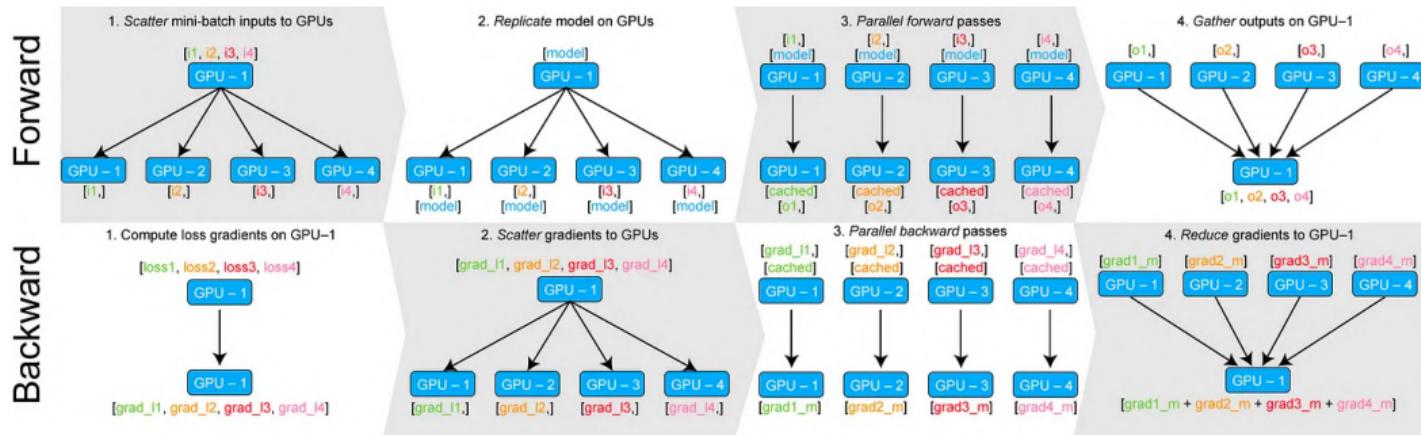


Масштабирование обучения

- ▶ При обучении память GPU в основном тратится на
 - ▶ веса модели
 - ▶ состояние оптимизатора
 - ▶ активации
 - ▶ градиенты
- ▶ Объем памяти GPU сильно ограничен: A100 имеет 80Гб
- ▶ У современных моделей даже веса могут не влезать в такой объём
- ▶ При обучении с помощью стандартного AdamW состояние оптимизатора требует $x2$ от размера модели
- ▶ Для обучения больших моделей с адекватной скоростью требуется обрабатывать большой объём данных одновременно
- ▶ Это возможно только при использовании параллельных вычислений на множестве GPU, этот процесс можно организовать по-разному

Масштабирование обучения

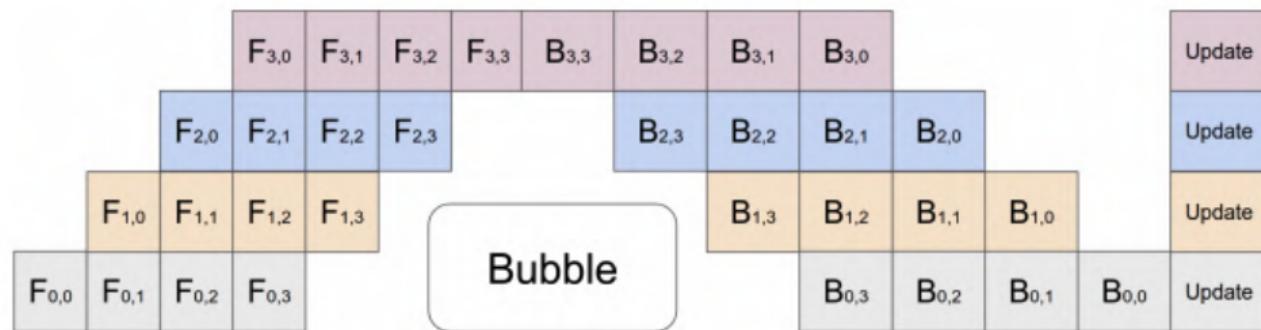
- ▶ Если модель и состояние оптимизатора не занимают всю память GPU, то очевидный способ параллелизма — по данным (Data Parallelism)
- ▶ Каждая GPU имеет свою копию модели и обрабатывает часть батча



- ▶ Можно добавить **Gradient Accumulation**: разделять батч по всем GPU не целиком, а частями для экономии памяти
- ▶ Градиенты агрегируются до обработки всего батча, после чего запускается обновление параметров модели

Масштабирование обучения

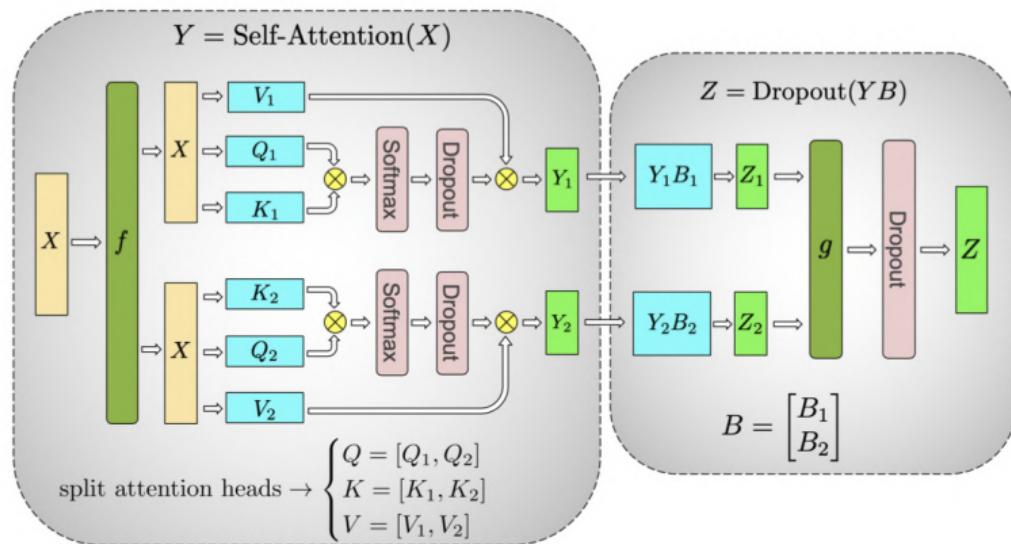
- ▶ Если памяти одной GPU не хватает, модель можно разрезать и разложить на несколько устройств
- ▶ Pipeline Parallelism — группы слоёв раскладываются по своим GPU
- ▶ Для уменьшения простоя батч нарезается на части, и более глубокие слои начинают работать раньше



- ▶ По сравнению с другими подходами требует сильно большего переписывания кода

Масштабирование обучения

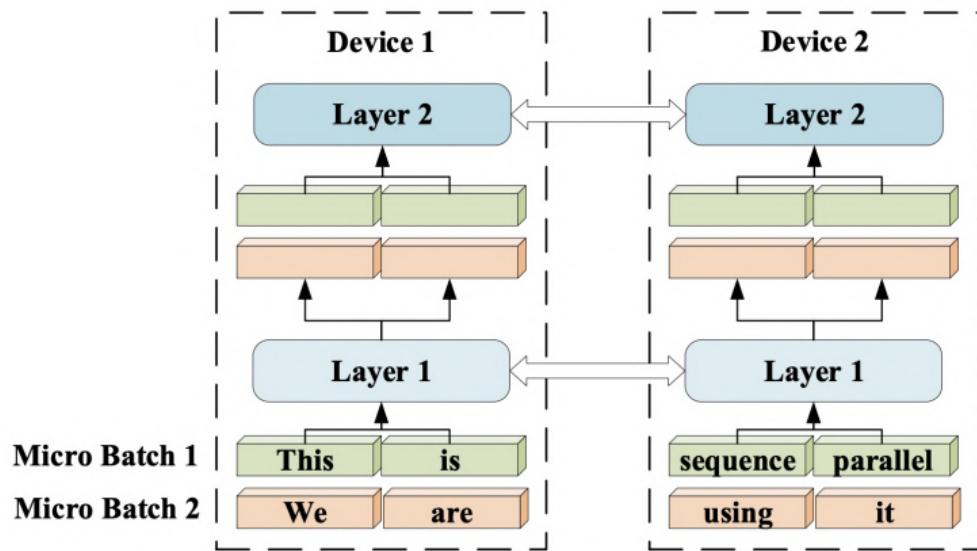
- Tensor Parallelism: по GPU раскладываются части тензора



- Self-attention параллелируется естественно за счёт разных голов
- При TP сетевые коммуникации более интенсивные, чем при DP или PP
⇒ модель лучше раскладывать на одном узле DGX или в сети InfiniBand

Масштабирование обучения

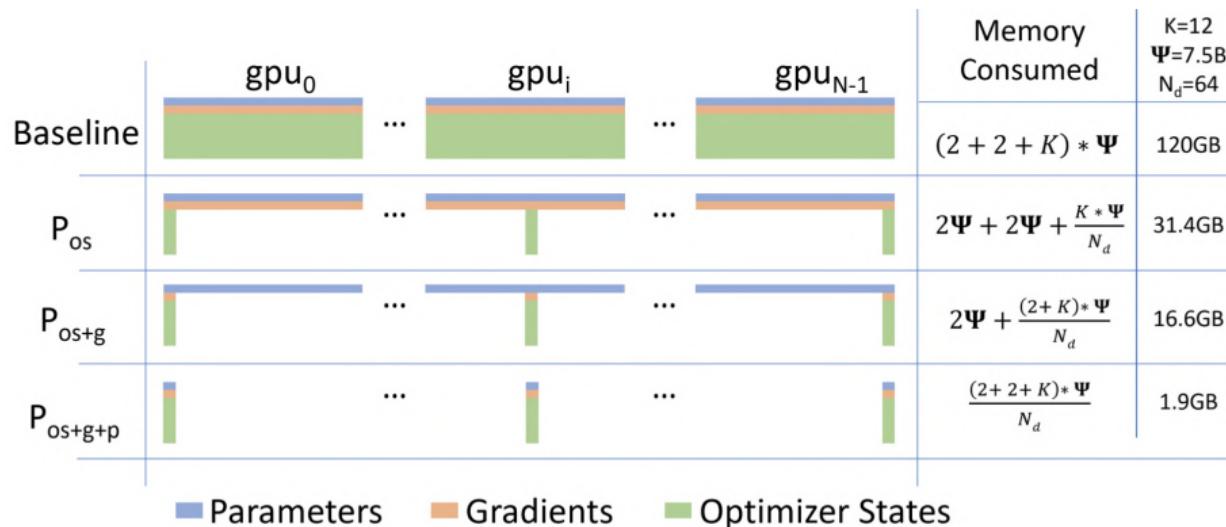
- ▶ Sequence Parallelism: по GPU раскладываются части последовательности



- ▶ Может естественно комбинироваться с Ring Attention (будет дальше)
- ▶ В определённых случаях эффективнее (за счёт меньшего объёма коммуникаций), чем TP, но может со всем комбинироваться

Масштабирование обучения

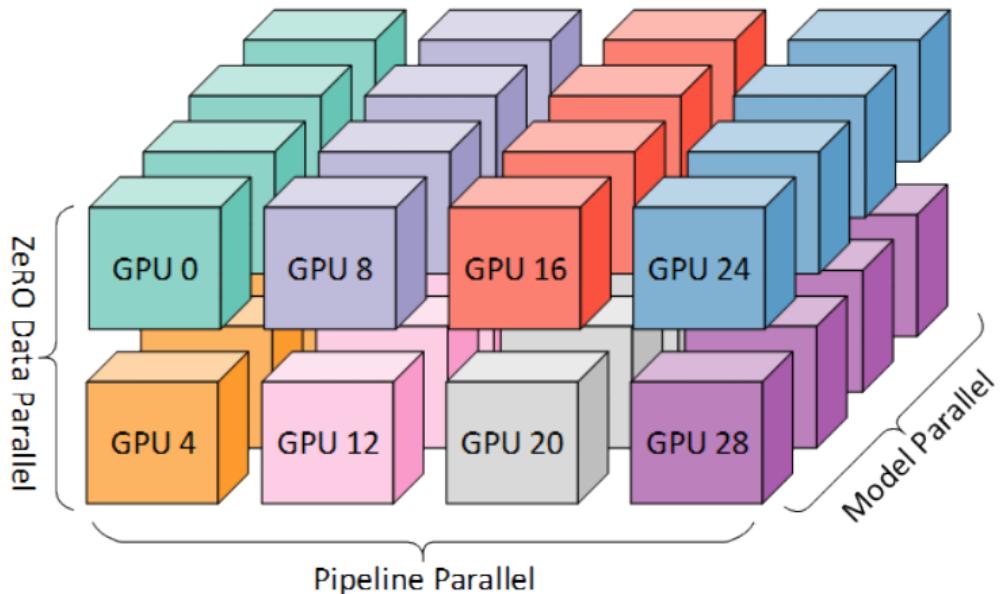
- ▶ В Data Parallelism можно избежать хранения избыточной информации с помощью [ZeRO](#), 2019, стандартная реализация — [DeepSpeed](#)
- ▶ На каждом этапе (stage) потребление падает, сетевые коммуникации растут x1.5 только на этапе 3



- ▶ ZeRO умеет выгружать данные в RAM, что тоже экономит память GPU

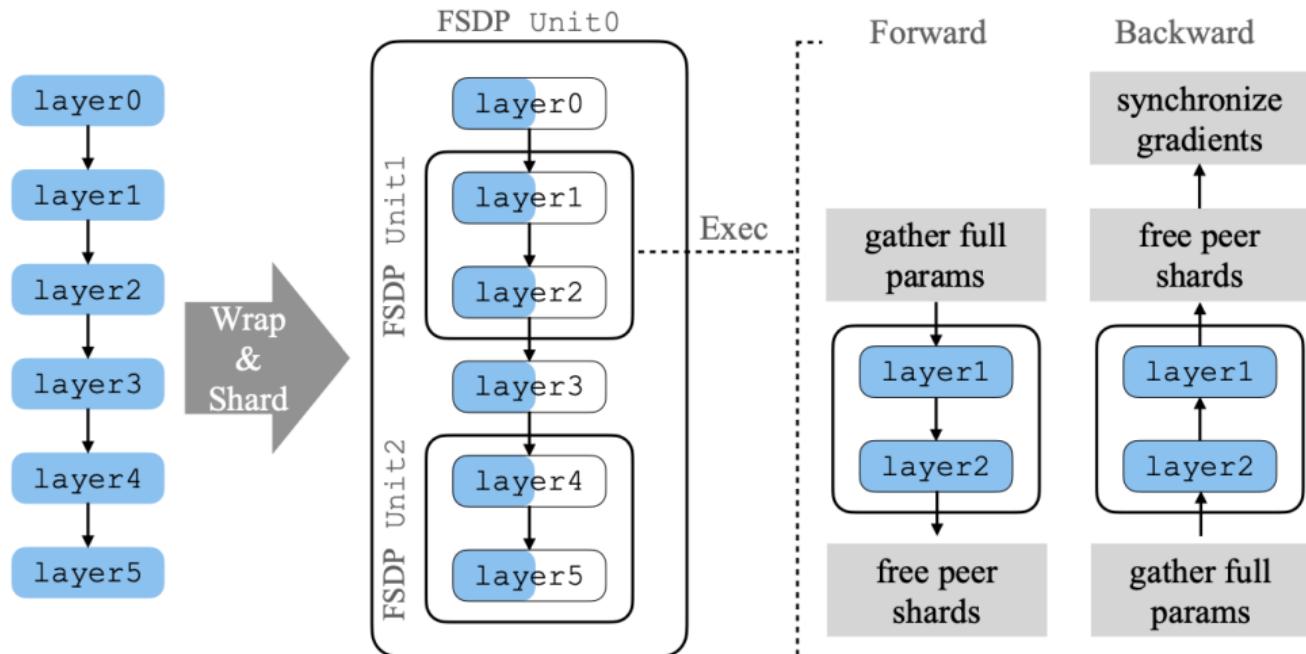
Масштабирование обучения

- ▶ Все техники могут применять как сами по себе, так и в комбинациях
- ▶ DP+PP+TP дают 3D-параллелизм, часто комбинируется с ZeRO stage 1 (stage 2/3 тоже можно, но сложнее + растут сетевые коммуникации)



Масштабирование обучения

- Более современная альтернатива DeepSpeed — FSDP
- Модель делится на юниты, каждый юнит раскладывается на группу GPU



Масштабирование обучения

- ▶ Раскладывать можно на все GPU (Full Sharding) или на часть (Hybrid), если видеокарт много относительно размера модели и микробатча

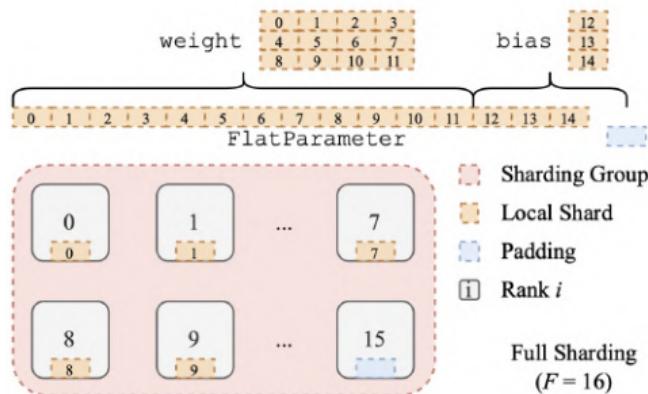


Figure 3: Full Sharding Across 16 GPUs

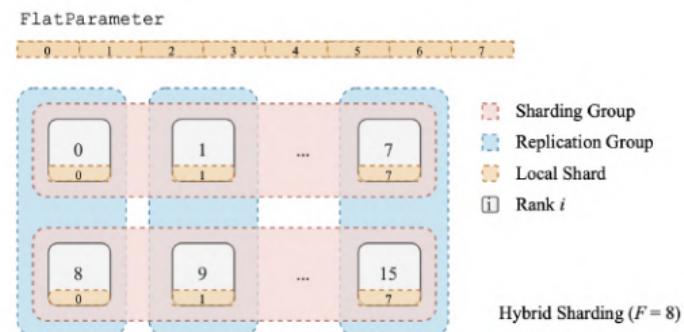
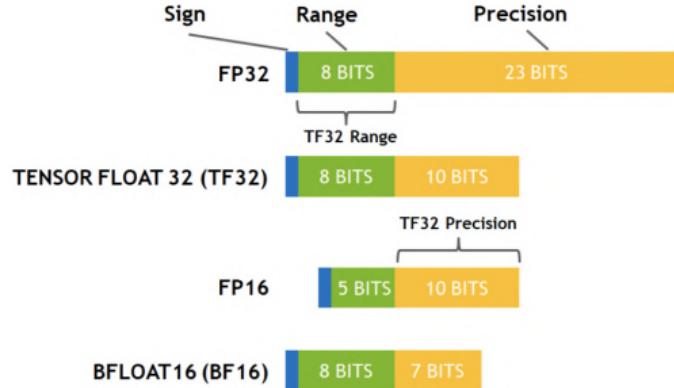


Figure 4: Hybrid Sharding on 16 GPUs: GPUs are configured into 2 sharding groups and 8 replication groups

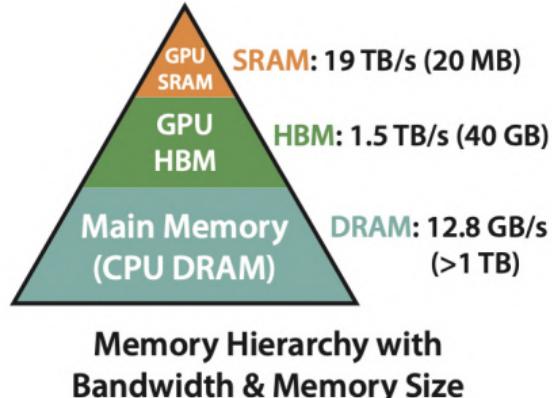
- ▶ Фреймворк FSDP активно используется, он эффективный и легко встраиваемый
- ▶ Если модель не влезает в память GPU, можно комбинировать с ТР и РР

Эффективность обучения

- ▶ Обучать модели в fp32 неэффективно
- ▶ Используют Mixed Precision:
 - ▶ две копии весов, в fp32 и fp16 / bf16 (если GPU Ampere)
 - ▶ активации считаются в fp16 / bf16
 - ▶ агрегации и нормализации в fp32
 - ▶ градиенты и состояние оптимизатора в fp32
- ▶ Затраты памяти нивелируются большим батчом, а обучение ускоряется
- ▶ fp16 требует масштабирования loss для стабильности (умножение на коэффициент и обратно), bf16 — нет, он в целом более стабилен
- ▶ GPU Ampere могут заменять fp32 на tf32 — более эффективный и экономичный формат, можно комбинировать это с Mixed Precision

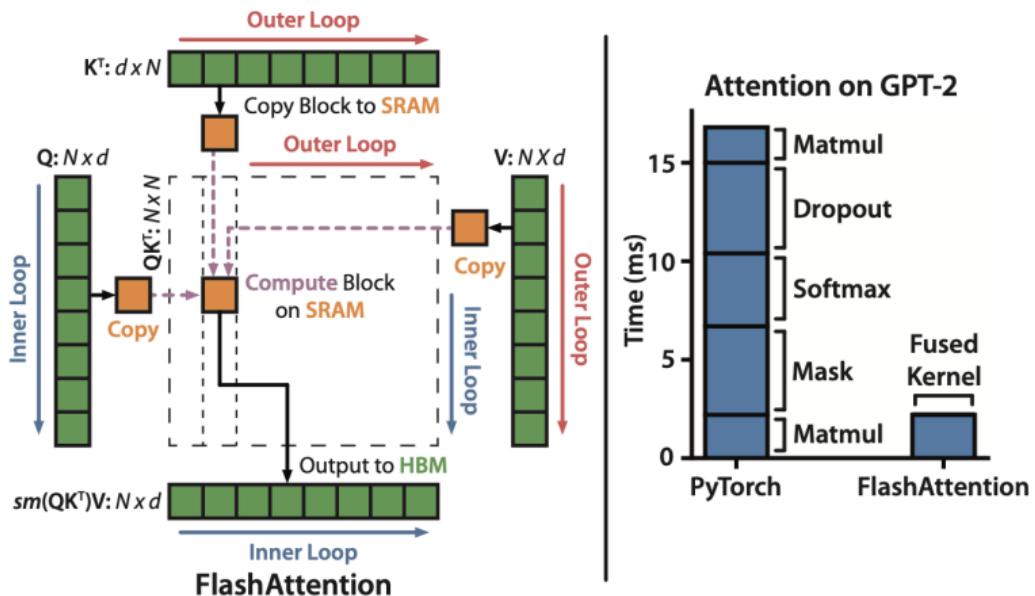


Эффективность обучения

- ▶ Утилизация GPU была очень неоптимальной — много времени уходило не на вычисления, а на пересылку данных между HBM и SRAM
 - ▶ Подсчёт softmax в self-attention генерирует промежуточные матрицы, которые занимают место и перемещаются
 - ▶ Можно ввести дополнительные переменные и считать softmax блочно
 - ▶ Не нужно хранить промежуточные матрицы, передача данных между HBM и SRAM становится экономичнее
 - ▶ Нужные для backward-шага промежуточные значения можно эффективно пересчитывать вместо хранения на forward
- 
- The diagram illustrates the memory hierarchy of a GPU system, represented as a triangle divided into three horizontal layers. The top layer is orange and labeled 'GPU SRAM'. The middle layer is green and labeled 'GPU HBM'. The bottom layer is teal and labeled 'Main Memory (CPU DRAM)'. To the right of the triangle, specific details are provided for each layer: 'SRAM: 19 TB/s (20 MB)' for the top layer, 'HBM: 1.5 TB/s (40 GB)' for the middle layer, and 'DRAM: 12.8 GB/s (>1 TB)' for the bottom layer. Below the triangle, the text 'Memory Hierarchy with Bandwidth & Memory Size' is centered.

Эффективность обучения

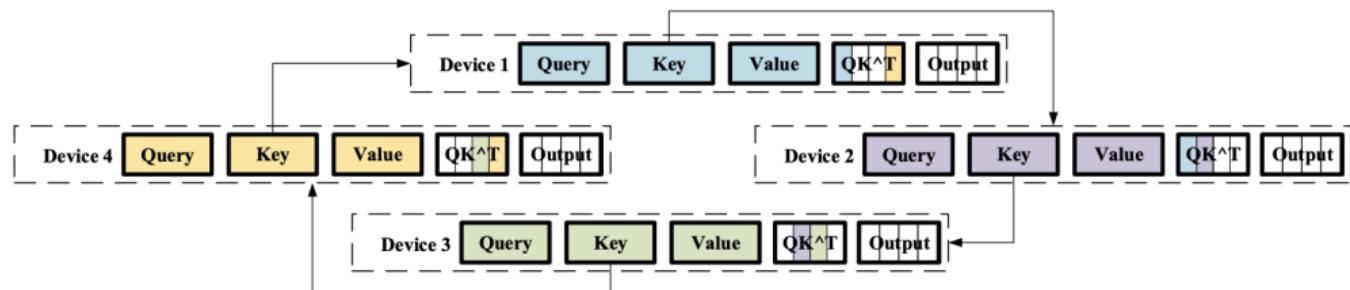
- Дополнительное ускорение получено за счёт **Fusing** — выполнения набора операций одним CUDA ядром



- **Flash Attention 2**, 2023 даёт ещё больший выигрыш по скорости за счёт вычислительных оптимизаций на GPU

Эффективность обучения

- ▶ Метод масштабирования точного подсчёта внимания на множество GPU для увеличения длины последовательности — [Ring Attention](#)
- ▶ Запросы Q , ключи K и значения V разбиваются на блоки по числу GPU и распределяются по ним
- ▶ Вычисления self-attention блочное, на каждой GPU:
 - ▶ для блока Q вычисляются все коэффициенты с текущим блоком K
 - ▶ блок K передаётся на следующую GPU, а текущая получает новый блок с предыдущей карты, и вычисление повторяется
 - ▶ после полного круга подсчёт идёт для V и агрегируется полный attention

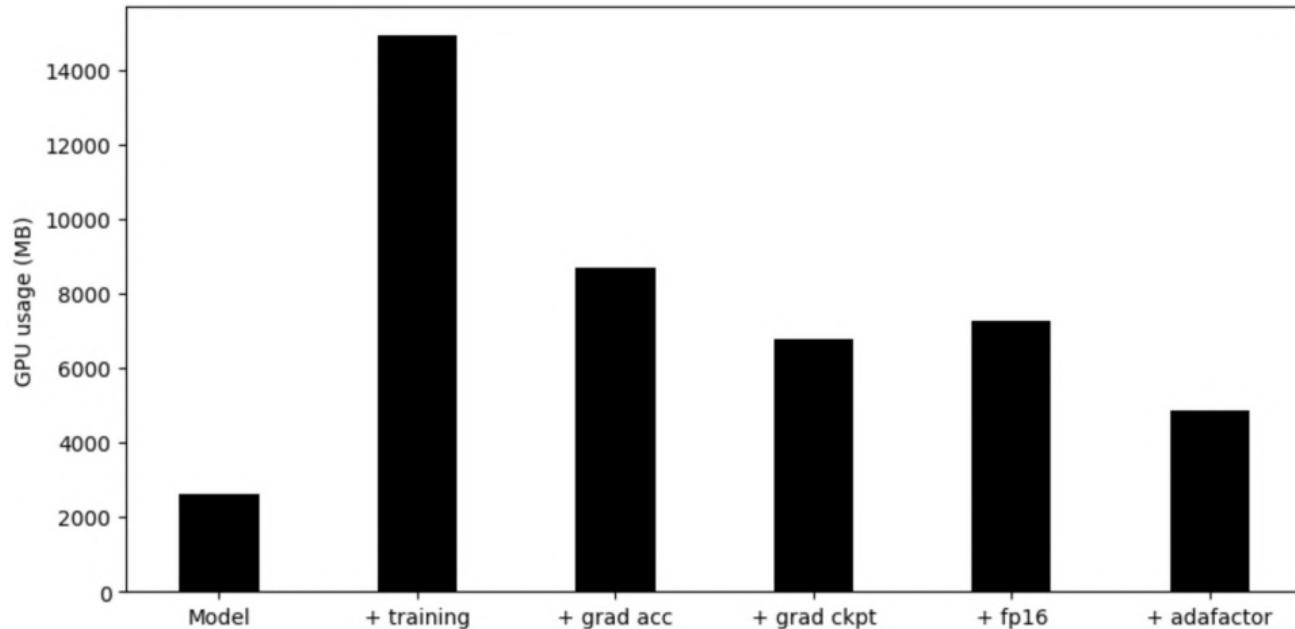


Эффективность обучения

- ▶ Важная техника для оптимизации памяти на полносвязных слоях — **Gradient (Activation) Checkpointing**
- ▶ Выходы каждого линейного слоя на forward-шаге нужны для вычисления градиентов на этом слое на обратном шаге, поэтому они сохраняются
- ▶ Это приводит к линейному по числу слоёв росту потребления памяти
- ▶ Можно ничего не хранить и вычислять для каждого слоя активации с нуля (т.е. от начала сети до этого слоя)
- ▶ Это экономит память, но объем вычислений на forward из линейного по числу слоёв становится квадратичным
- ▶ **Решение:** сохранять активации части слоёв на некотором расстоянии друг от друга (*checkpoint*)
- ▶ Вычисление активаций слоя будет идти от последнего чекпойнта
- ▶ В среднем потребление памяти падает с $O(n)$ до $O(\log n)$ за счёт замедления примерно на 20%

Эффективность обучения

- ▶ Можно экспериментировать с оптимизаторами, например, [Adafactor](#) более экономичный по памяти, чем AdamW
- ▶ Большинство оптимизаций хорошо комбинируются друг с другом:



Алгоритмы оптимизации

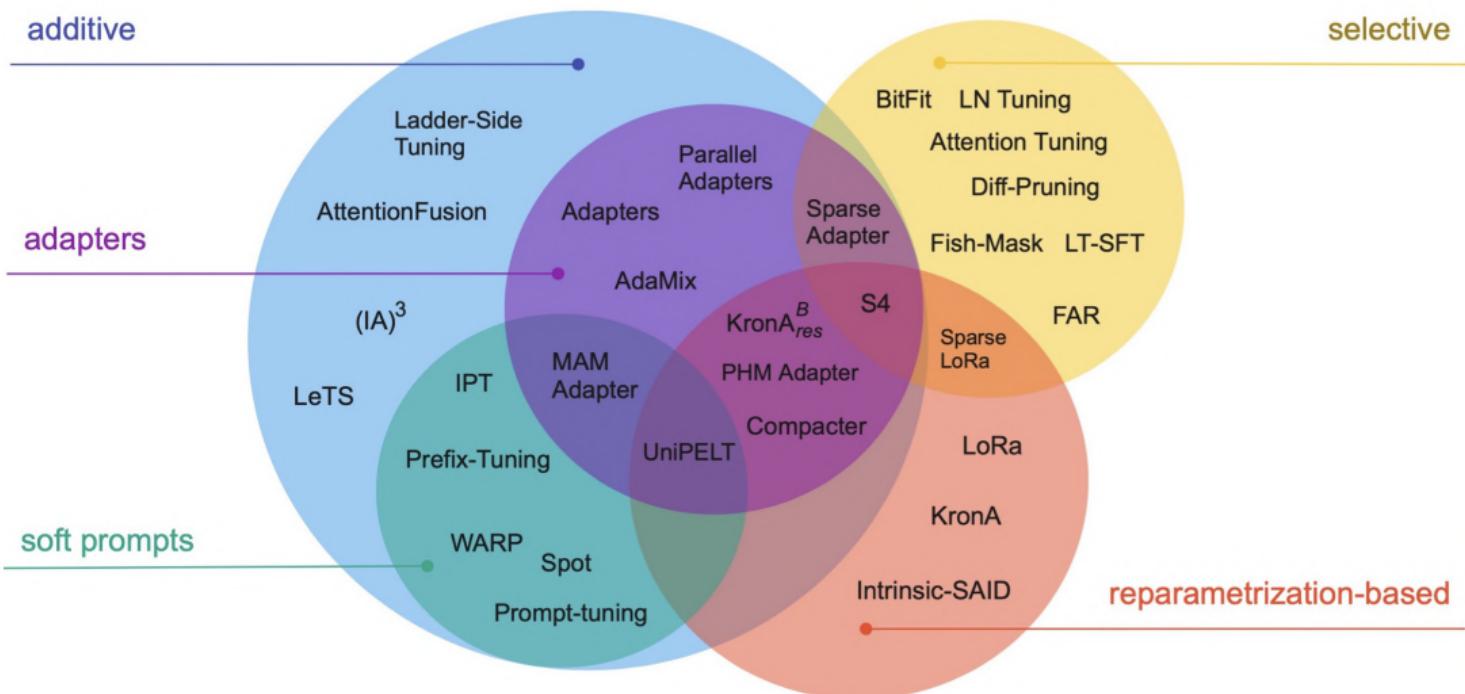
- ▶ Стандартный алгоритм обучения LLM — [AdamW](#), 2017:
 - ▶ в основе лежит [Adam](#), 2014 ([Momentum](#), 1986 + [RMSprop](#), 2012)
 - ▶ моменты 1-го и 2-го порядков считаются, хранятся и используются на шаге обновления весов
 - ▶ в отличие от Adam, AdamW делает weight decay регуляризацию на параметрах, а не на градиентах (работает лучше, чем L2 на loss)
- ▶ Популярные альтернативы:
 - ▶ [Adafactor](#), 2018
 - ▶ [Shampoo](#), 2018
 - ▶ [Adan](#), 2018
 - ▶ [AMSGrad](#), 2019
 - ▶ [Sophia](#), 2023
 - ▶ [Lion](#), 2023
- ▶ Полноценно обойти AdamW сложно — для прочих алгоритмов слишком мало разносторонних экспериментов, проводить их долго и дорого
- ▶ Для уменьшения потребления памяти используются квантованные варианты алгоритмов (например, [8-bit AdamW](#), 2021)

Эффективность дообучения

- ▶ Стандартный подход в использовании LLM — Transfer Learning
- ▶ Большая и умная модель адаптируется под частные задачи с помощью дообучения на небольшом наборе данных
- ▶ **Проблема:** дообучение LLM целиком может требовать больших ресурсов и времени
- ▶ **Возможное решение:** учить не всю модель, а только отдельные слои
- ▶ **Проблема:** задачи могут быть многочисленными и разнообразными — не хочется на каждую учить, хранить и хостить целую модель
- ▶ Альтернатива полному или частичному дообучению — [адаптеры](#)
- ▶ Модель остаётся неизменной, к ней как-то добавляются немного новых параметров (или используется малая часть исходных весов)
- ▶ При дообучении эти параметры настраиваются корректировать работу модели для повышения качества на целевой задаче

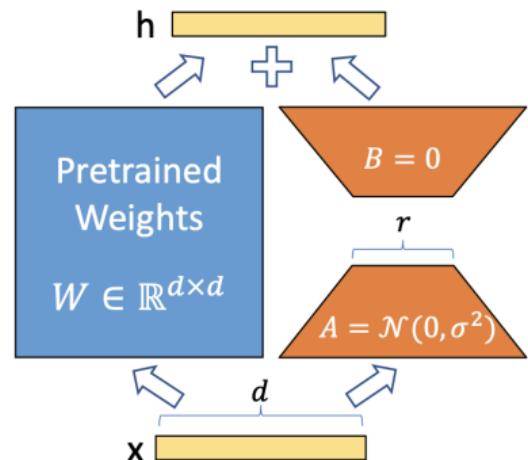
Эффективность дообучения

► Адаптеров придумали очень много:



Эффективность дообучения

- Одним из наиболее популярных методов остаётся LoRA:
 - веса модели полностью замораживаются, выбираются целевые линейные веса
 - для каждой матрицы весов заводится пара новых матриц — её низкоранговое разложение
 - при работе эти матрицы перемножаются и результат складывает с основными замороженными весами
 - хороший рецепт: добавлять адаптер на все матрицы весов запросов и значений в self-attention



Качество генерации

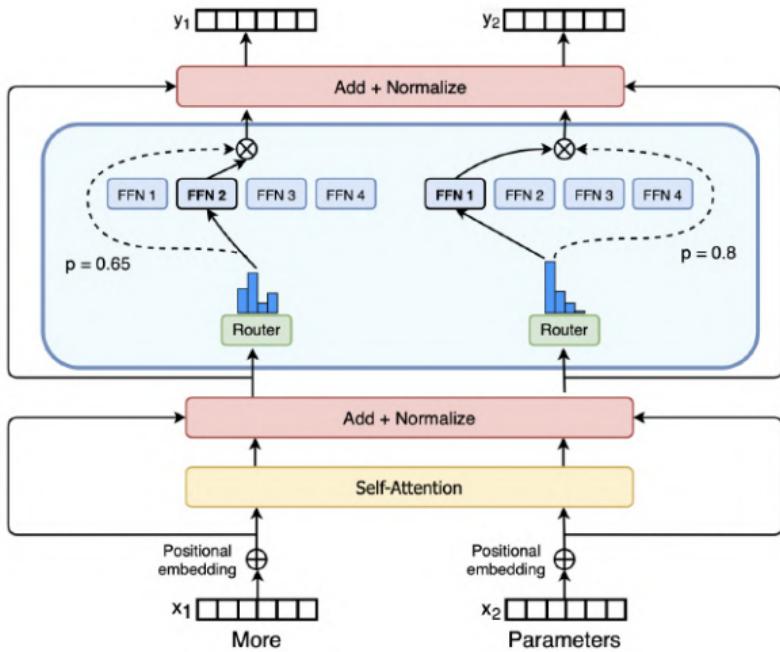
- ▶ Техника на основе идей ансамблирования: [Checkpoint Averaging](#) — усреднение весов нескольких версий модели в конце обучения
- ▶ Выбор и настройка **метода декодирования** выходов LLM:
 - ▶ Greedy search
 - ▶ Beam search (num_beams)
 - ▶ Top-K sampling (top_k)
 - ▶ Top-P [nucleus] sampling (top_p)
 - ▶ Contrastive search (top_k, penalty_alpha)
- ▶ Неочевидная проблема — выбор токенов на границе промпта и ответа:

```
link is <a href="http: -> link is <a href="http: //site.com  
link is <a href="http -> link is <a href="http://site.com
```

- ▶ Возможное решение — [Token Healing](#): до генерации откатиться на один или более токенов назад и начать оттуда

Архитектура

- ▶ Если учить оптимально, то больше весов \Rightarrow выше качество
- ▶ Но больше весов \Rightarrow медленнее инференс
- ▶ Решение — Mixture-of-Experts, «эксперты» FF-слои
- ▶ Вектор слова после self-attention идёт Router, тот отправляет в его к лучшим экспертам
- ▶ Параметров много, но при обработке активируется только малая часть
- ▶ Каждый эксперт учится решать свои типы задач



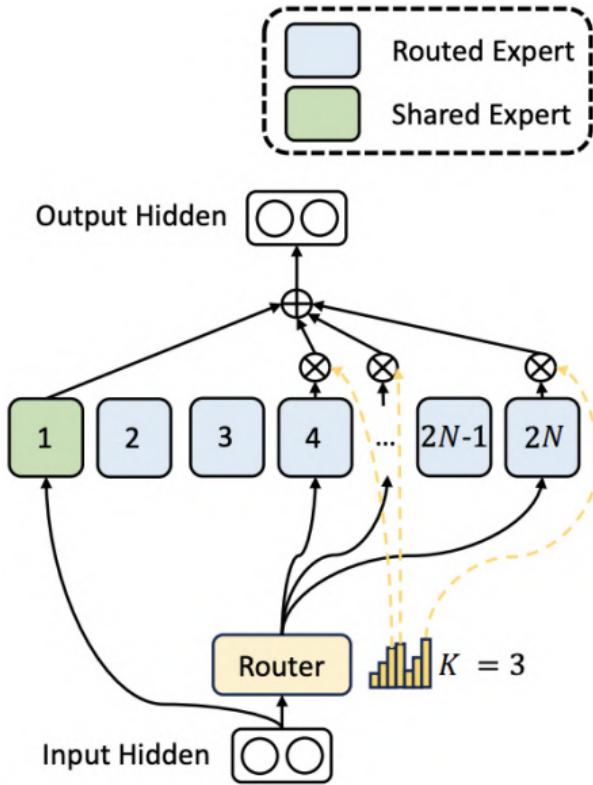
Архитектура

- ▶ У MoE есть проблемы, из-за которых идея не сразу стала популярной:
 - ▶ неравномерность загрузки экспертов (нужен loss на Router для балансировки + ограничение на capacity эксперта)
 - ▶ более низкое качество и проблемы с дообучением (сильно помогло использование LLaMA-like архитектуры)
- ▶ Важно: исходная идея MoE — эксперты с разными доменными знаниями
- ▶ Вместо этого эксперты улавливают родственные токены (имена, артикли)

Punctuation	Layer 2	,.,,-.,.).)
	Layer 6	,,,,:.,&,&&?&-,,?,,,<extra_id_27>
Conjunctions and articles	Layer 3	The
	Layer 6	a and and and and and and or and a and . the the if ? a designed does been is not
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died falling designed based disagree submitted develop

Архитектура

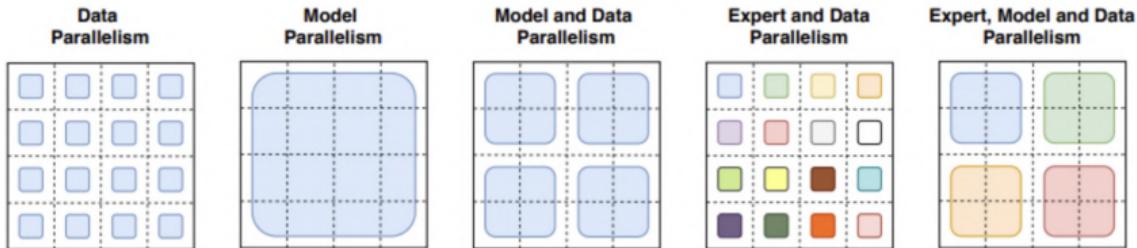
- ▶ В DeepSeekMoE предлагается решение:
 - ▶ вектор токена разделяется на части, каждая идёт своему эксперту
 - ▶ экспертов становится больше, у каждого меньшая размерность
 - ▶ добавляются общие эксперты, в которые части токена попадают всегда
- ▶ При том же объёме вычислений и числе параметров качество и скорость инференса выше, чем у обычного MoE
- ▶ Эксперты более доменные, т.е. важные — удаление даже нескольких ощутимо роняет perplexию



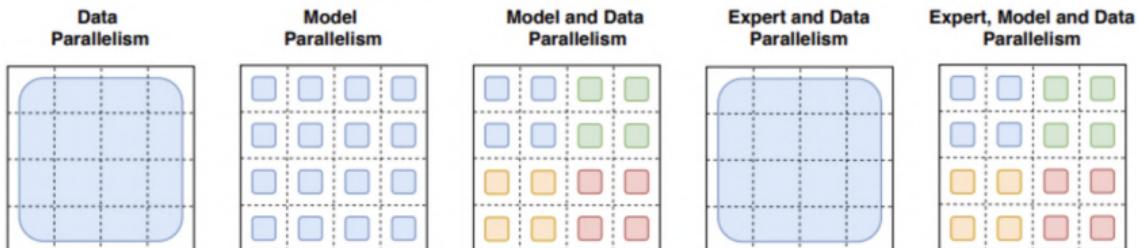
Масштабирование обучения

- ▶ Для моделей с MoE можно использовать Expert Parallelism
- ▶ Его можно комбинировать с другими типами параллелизма:

How the *model weights* are split over cores

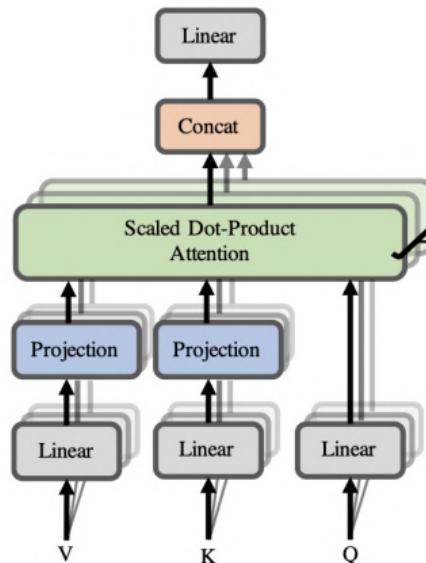


How the *data* is split over cores



Обработка длинного контекста

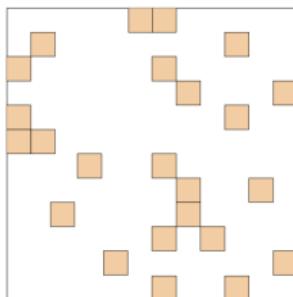
- ▶ Для обхода квадратичной сложности self-attention можно **понизить сложность вычислений** за счёт **понижения размерностей** матриц



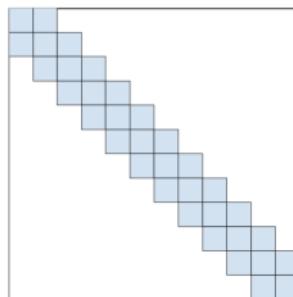
- ▶ Примеры работ:
 - ▶ Linformer: Self-Attention with Linear Complexity, 2020
 - ▶ Rethinking Attention with Performers, 2020

Обработка длинного контекста

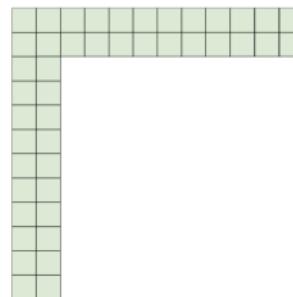
- ▶ Для обхода квадратичной сложности self-attention можно **вычислять внимание по частям последовательности**



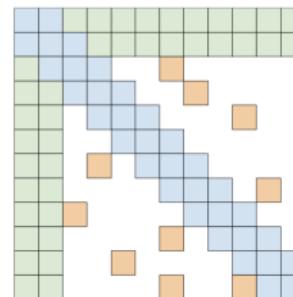
(a) Random attention



(b) Window attention



(c) Global Attention

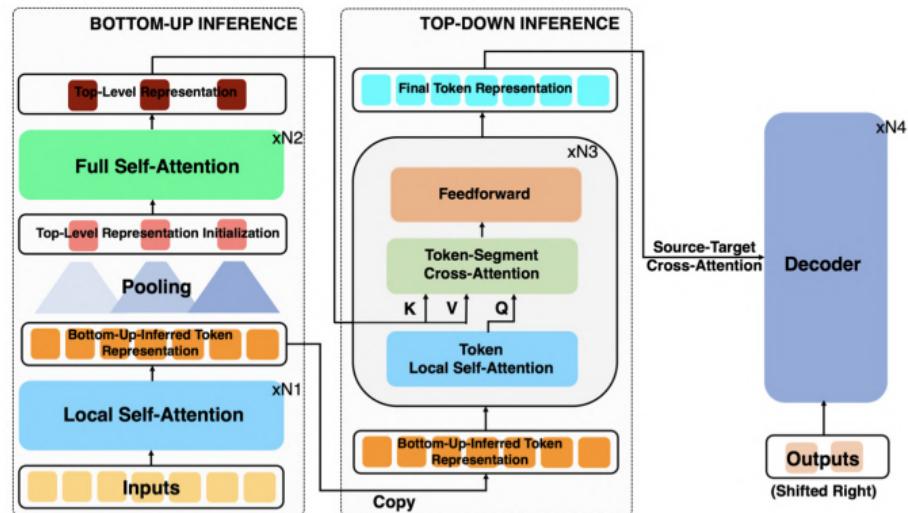


(d) BIGBIRD

- ▶ Примеры работ:
 - ▶ Generating Long Sequences with Sparse Transformers, 2019
 - ▶ Longformer: The Long-Document Transformer, 2020
 - ▶ Big Bird: Transformers for Longer Sequences, 2020
 - ▶ LongT5: Efficient Text-To-Text Transformer for Long Sequences, 2021
 - ▶ LongNet: Scaling Transformers to 1,000,000,000 Tokens, 2023

Обработка длинного контекста

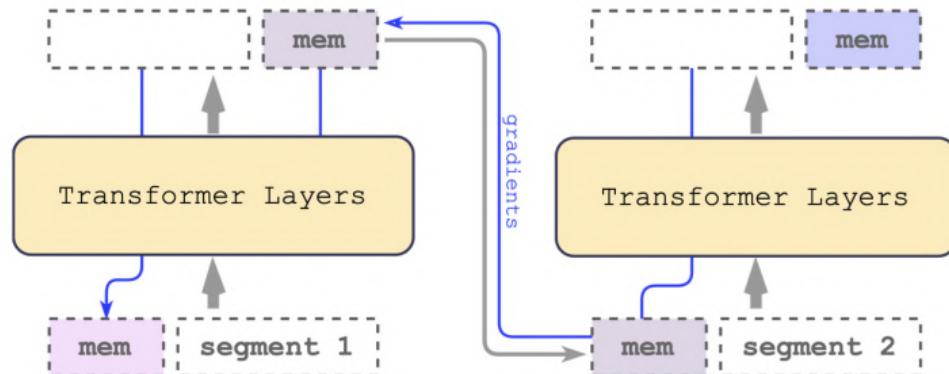
- ▶ Для обхода квадратичной сложности self-attention можно обрабатывать последовательность иерархически



- ▶ Примеры работ:
 - ▶ Long Document Summarization with Top-down and Bottom-up Inference, 2022
 - ▶ Efficient Long-Text Understanding with Short-Text Models, 2022

Обработка длинного контекста

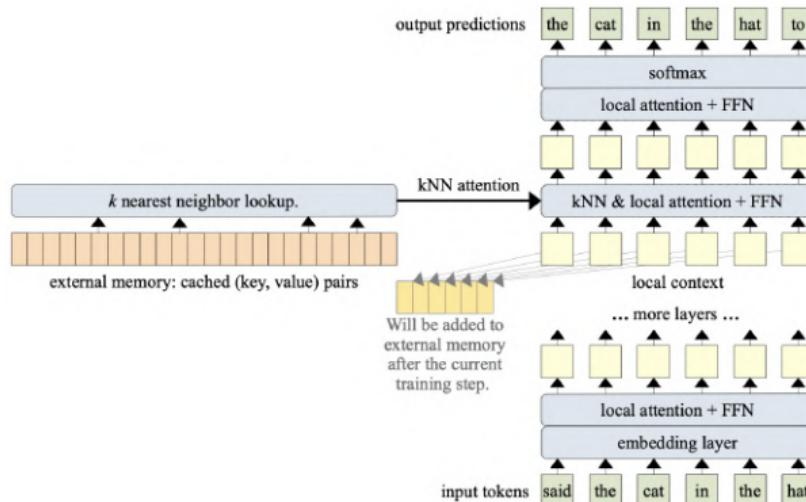
- ▶ Для обхода квадратичной сложности self-attention можно вместо увеличения длины последовательности передавать контекст рекуррентно



- ▶ Примеры работ:
 - ▶ Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, 2019
 - ▶ Scaling Transformer to 1M tokens and beyond with RMT, 2023

Обработка длинного контекста

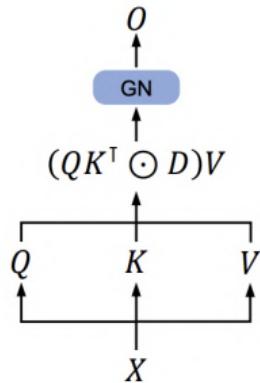
- ▶ Для обхода квадратичной сложности self-attention можно вместо увеличения длины последовательности сохранять контекст в kNN-индексе



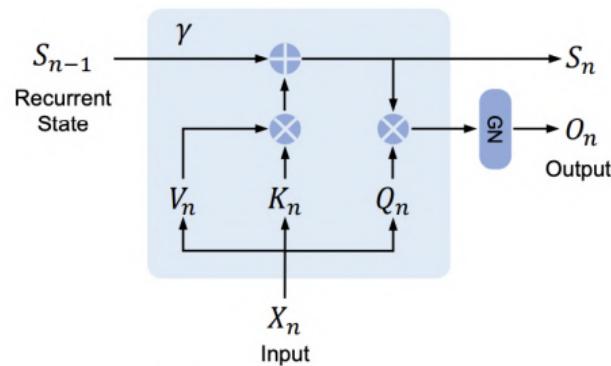
- ▶ Примеры работ:
 - ▶ Memorizing Transformers, 2022
 - ▶ Unlimiformer: Long-Range Transformers with Unlimited Length Input, 2023
 - ▶ Focused Transformer: Contrastive Training for Context Scaling, 2023

Обработка длинного контекста

- ▶ Для обхода квадратичной сложности self-attention можно заменить внимание на другой механизм, схожий по качеству и скорости обучения, и вычисляемый на инференсе рекуррентно (варианты SSM)



(a) Parallel representation.

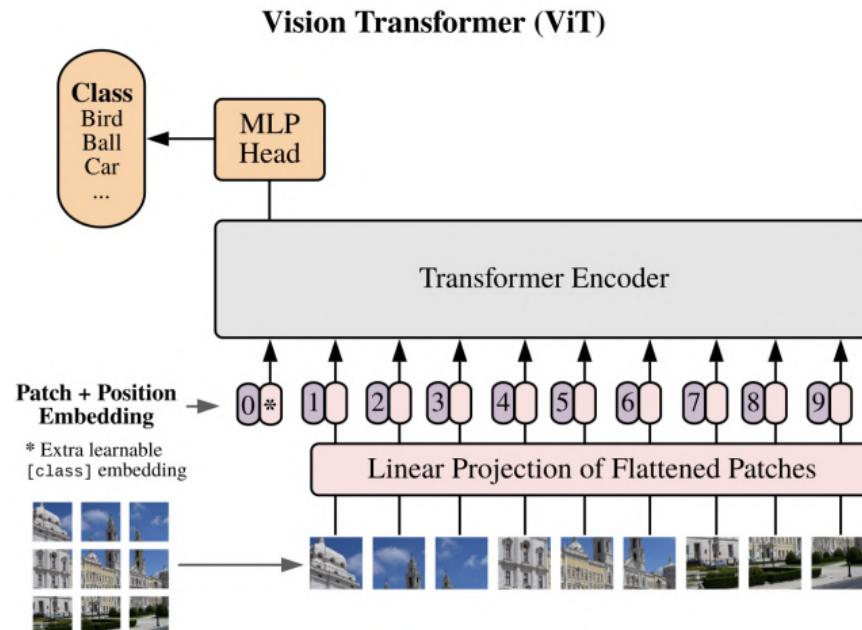


(b) Recurrent representation.

- ▶ Примеры работ:
 - ▶ RWKV: Reinventing RNNs for the Transformer Era, 2023
 - ▶ Retentive Network: A Successor to Transformer for Large Language Models, 2023
 - ▶ Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2023

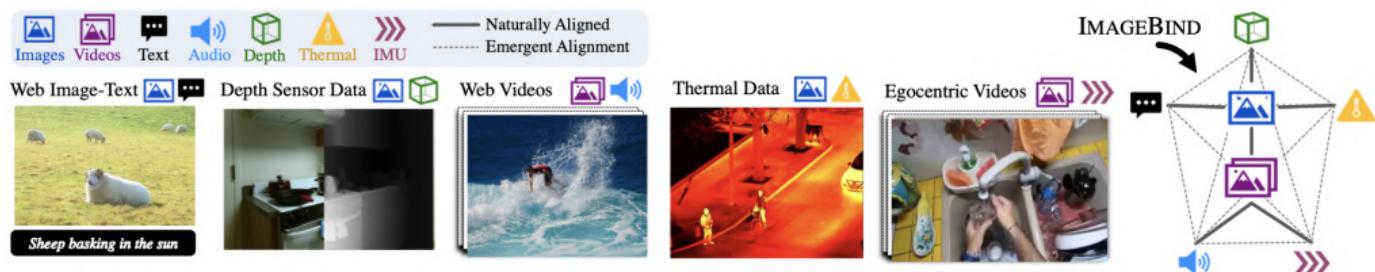
Мультимодальность

- ▶ Языковые модели можно использовать для понимания не только текста, но других модальностей: изображения, видео, аудио
- ▶ Популярный [Vision Transformer, 2020](#), активно используется в качестве кодировщика изображений, похож на BERT:



Мультимодальность

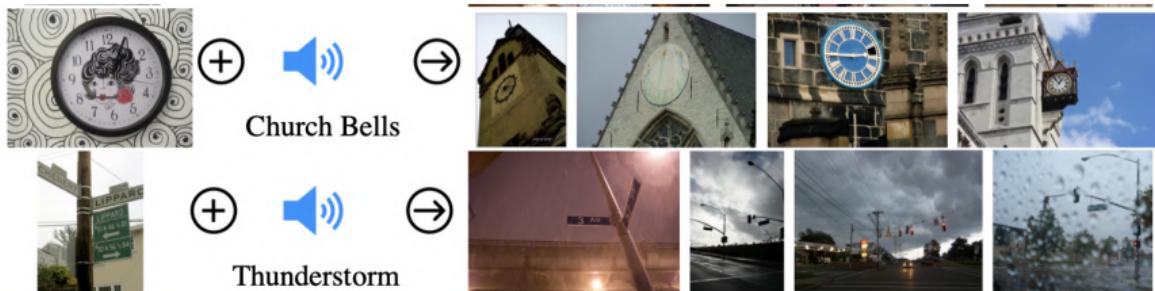
- ▶ Объекты разных модальностей можно погрузить в одно векторное пространство
- ▶ Можно собрать попарные данные между всеми модальностями — дорого и сложно
- ▶ А можно использовать в качестве связующего звена изображения, как в [ImageBind](#), 2023:



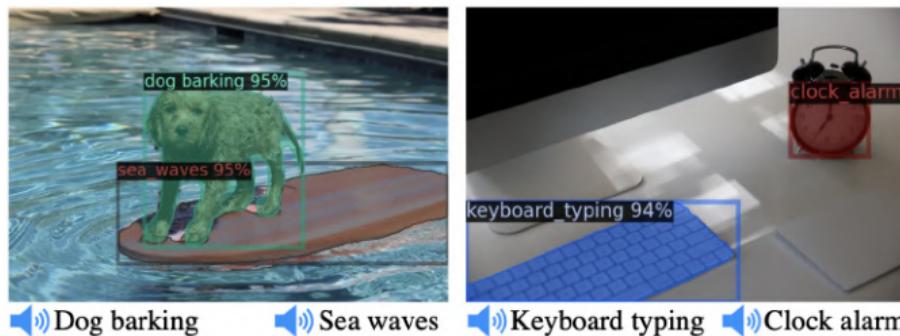
- ▶ Полученные векторы можно доучивать для передачи в LLM (например, [NExT-GPT](#) и [ImageBind-LLM](#), 2023)

Мультимодальность

- ▶ В итоговом пространстве можно складывать векторы разных модальностей:

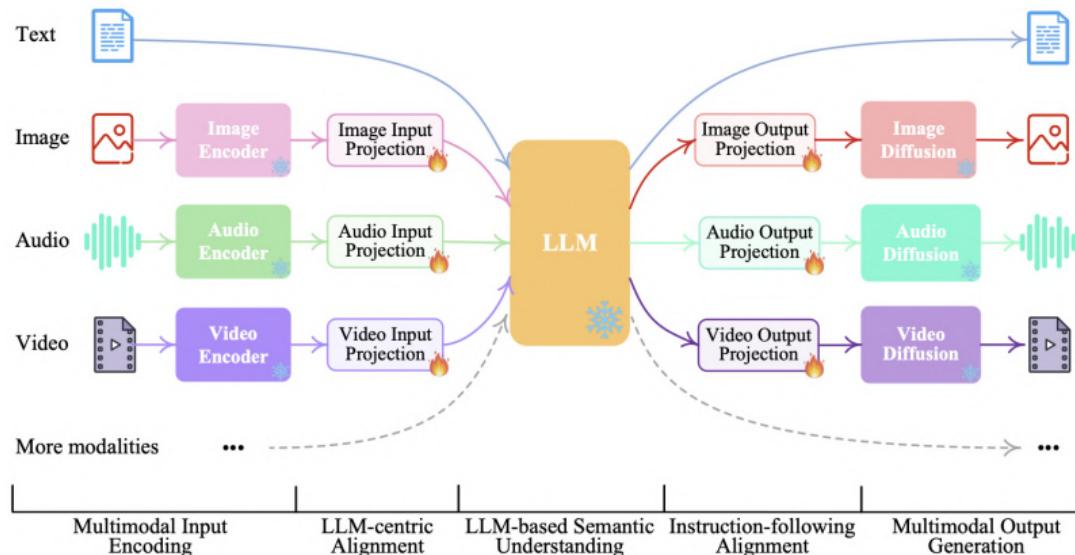


- ▶ Внешней модели сегментации с векторизатором (CLIP) на входе можно подсунуть вектор аудио:



Мультимодальность

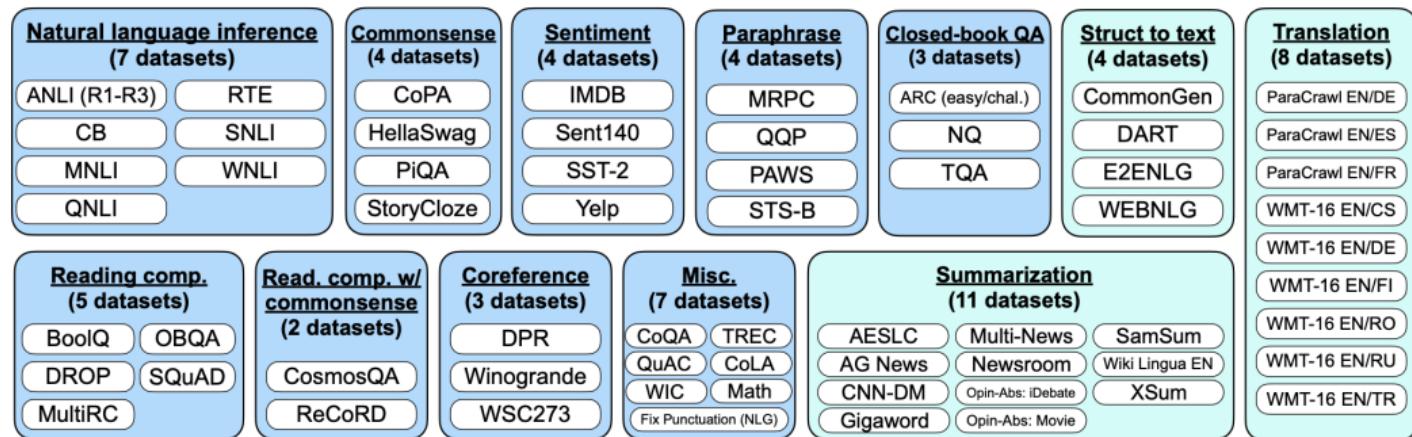
- LLM над общими векторами с диффузионными генераторами дают системы, принимающие и генерирующие объекты разных модальностей (NExT-GPT):



- В таких случаях тяжёлые модели обычно заморожены, а учатся проекционные слои и адаптеры

Оценка качества LLM

- ▶ Способности моделей проверяются путём решения разных текстовых или мультимодальных задач на разных наборах данных
- ▶ Схема различных NLP-задач и соответствующих данных (синие — с короткими ответами, бирюзовые — с длинными):



Оценка качества LLM

- ▶ Из отдельных наборов данных формируют коллекции — бенчмарки
- ▶ Бенчмарков много для разных задач, длины контекста, доменов, языков и модальностей, примеры популярных:
 - ▶ [MMLU](#), 2020 (тексты, английский)
 - ▶ [HumanEval](#), 2021 (программный код, английский)
 - ▶ [MMBench](#), 2023 (тексты и изображения, английский)
 - ▶ [LongBench](#), 2023 (длинные тексты, английский)
 - ▶ [MERA](#), 2023 (тексты, русский)
- ▶ Проверка коротких ответов автоматическая, с длинными сложнее —
автометрики слабые, проверяют люди или более сильные LLM (GPT-4):
[«Ты выступаешь в роли асессора. Тебе покажут правильный пересказ текста и пересказ, сгенерированный моделью, твоя задача оценить по шкале от 1 до 10 качество генерации пересказа ...»](#)

Спасибо за внимание!



Мурат Апишев

Search Tech Lead, Samokat.Tech
ex-Lead Data Scientist, SberDevices
mel-lain@yandex.ru